**PROIECT**

**LANT DE FARMACII**

**Samoila Alexandra**

**Grupa 30238**

**CUPRINS**

1. **Specificarea cerințelor 3**
2. **Instrumente utilizate 7**
3. **Etapa de analiză 12**
   1. **Diagrama cazurilor de utilizare 13**
   2. **Diagrame de activități 14**
4. **Etapa de proiectare 24**
   1. **Diagrame de clase 24**
      1. **Șablonul arhitectural Client-Server 25**
      2. **Șabloane de proiectare 28**
   2. **Diagrame de secvență**
   3. **Baza de date 33**
5. **Manual de utilizare 35**
6. **Bibliografie**
7. **Specificarea cerințelor**

Cerinta temei 1:

Descrierea aplicației:

Dezvoltați (analiză, proiectare, implementare) o aplicație desktop care poate fi utilizată într-un lanț de farmacii / plafare.

Aplicația va avea două tipuri de utilizatori: angajat al lanțului de farmacii / plafare și administrator.

Utilizatorii de tip angajat al lanțului de farmacii/plafare pot efectua următoarele operații după autentificare:

* Vizualizarea tuturor produselor dintr-o farmacie selectată / dintr-un plafar selectat;
* Filtrarea produselor după următoarele criterii: disponibilitate, valabilitate, preț, producător;
* Căutarea unui produs după denumire;
* Vizualizarea unor statistici legate de produse: procente după disponibilitate, termen de valabilitate, categorii, pret utilizând grafice (structură radială, structură inelară, de tip coloană, etc.).
* Operații CRUD în ceea ce privește persistența produselor;
* Salvare rapoarte / liste cu produse în mai multe formate: csv, xml, json.

Utilizatorii de tip administrator pot efectua următoarele operații după autentificare:

* Operații CRUD pentru informațiile legate de utilizatorii de tip angajat.

Constrângeri ale aplicației:

* Datele utilizate în aplicație vor fi socate într-un fișier binar.
* Utilizați șabloanele de proiectare creaționale Abstract Factory și Builder.

Cerinta temei 2 :

Obiectiv: Obiectivul acestei teme este familiarizarea cu șablonul architectural Client/Server.

Cerințe:

Transformați aplicația desktop implementată la tema 1 într-o aplicație client-server. Mare parte din modelul logic (inclusiv persistența) va face parte din server, mai puțin partea care poate fi determinată din datele primite de la server (generare rapoarte, statistică, etc.).

* În faza de analiză realizați diagrama cazurilor de utilizare.
* În faza de proiectare realizați diagrama de clase corespunzătoare aplicației server și diagrama de clase corespunzătoare aplicației client.
* În faza de implementare scrieți cod pentru îndeplinirea tuturor funcționalităților precizate de diagrama cazurilor de utilizare utilizând unul dintre următoarele limbaje de programare: C#, C++, Java, Python.
* Finalizarea temei va consta în predarea unui director ce va cuprinde:
* Un fișier word care cuprinde numele studentului, grupa, enunțul problemei și instrumentele utilizate.
* Un fișier cu diagramele UML realizate;
* Directorul cu aplicația implementată.

Constrângeri ale aplicației:

* Pentru persistență se va utiliza o bază de date relațională (MySQL, SQL Server, Oracle, etc.).
* Se cere utilizarea arhitecturii client/server, NU web client/server.

Observație: Se va utiliza baza de date doar în aplicațiile unde este necesară persistența.

Cerinta temei 3 :

Obiectiv: Obiectivul acestei teme este familiarizarea cu șablonul de proiectare comportamental Observer.

Cerințe: Optimizați aplicația client/server implementată la tema 2 astfel încât să utilizați șablonul de proiectare comportamental Observer în dezvoltarea aplicației client. În plus, interfața grafică a aplicației client va fi disponibilă în cel puțin 3 limbi de circulație internațională (implicit limba română).

* În faza de analiză realizați diagrama cazurilor de utilizare și diagramele de activități corespunzătoare tuturor cazurilor de utilizare.
* În faza de proiectare realizați:
* diagrama de clase corespunzătoare aplicației server;
* diagrama de clase corespunzătoare aplicației client;
* diagrame de secvență corespunzătoare tuturor cazurilor de utilizare.
* În faza de implementare scrieți cod pentru îndeplinirea tuturor funcționalităților precizate de diagrama cazurilor de utilizare utilizând unul dintre următoarele limbaje de programare: C#, C++, Java, Python.
* Finalizarea temei va consta în predarea unui director ce va cuprinde:
* Un fișier word care cuprinde numele studentului, grupa, enunțul problemei, instrumentele utilizate și detalii despre modul în care a fost utilizat șablonul de proiectare Observer.
* Un fișier cu diagramele UML realizate;
* Directorul cu aplicația implementată.

Observație: Pentru persistență se va utiliza o bază de date relațională (MySQL, SQL Server, Oracle, etc.). Se va utiliza baza de date doar în aplicațiile unde este necesară persistența.

1. **Instrumente utilizate**

Am ales sa lucrez in java deoarece acest limbaj imi place cel mai mult si il stiu cel mai bine, iar la vara as vrea sa fac un internship pe java si atunci acest proiect ma ajuta sa fiu mai bine pregatita.

***Java:*** 11

***Mediul de dezvoltare****:* Eclipse

***Versiunea mediului:*** 2019-12 (4.14.0)

In luna Mai, anul acesta, Java va împlini 25 de ani. Mai mult, când și-a aniversat ziua, limbajul de programare Java încă era (și este în continuare) în topul celor mai populare limbaje de programare.Chiar și după 25 de ani de la lansare, Java continuă să fie nu doar relevant, ci și influent.

**Java** este un**limbaj de programare OOP sau orientat-obiect**, dezvoltat de James Gosling la Sun Microsystems (acum filială Oracle), la începutul anilor ʼ90 și lansat în 1995.

Limbajul împrumută o mare parte din sintaxă de la C și C++, dar are un model al obiectelor mai simplu și prezintă mai puține facilități de nivel jos. Un program Java compilat, corect scris, poate fi rulat fără modificări pe orice platformă care e instalată o mașină virtuală Java (engleză Java Virtual Machine, prescurtat JVM). Acest nivel de portabilitate (inexistent pentru limbaje mai vechi cum ar fi C) este posibil deoarece sursele Java sunt compilate într-un format standard numit cod de octeți (engleză byte-code) care este intermediar între codul mașină (dependent de tipul calculatorului) și codul sursă.

Mediul de dezvoltare:

**Eclipse IDE 2019-12 (4.14.0)**

Eclipse este un mediu de dezvoltare open-source scris preponderent în Java. Acesta poate fi folosit pentru a dezvolta aplicații Java și, prin intermediul unor plug-in-uri, în alte limbaje, cum ar fi C, C++, COBOL, Python, Perl și PHP.

Eclipse este platforma extinsă a clientului (RCP - eng. rich client platform). Este compusă din următoarele componente:

* Nucleul platformei (încărcare Eclipse, module de lansare);
* OSGi (mediu standard de livrare);
* SWT (set de instrumente widget portabile);
* JFace (procesare de text, editoare de text);
* Eclipse mediu de lucru (panouri, editori, proiecții).

GUI în Eclipse este scris folosind setul de instrumente SWT. Acesta din urmă, spre deosebire de Swing (care emite independent controale grafice), utilizează componentele grafice ale sistemului de operare dat. Interfața de utilizator Eclipse depinde, de asemenea, de un strat intermediar GUI numit JFace, care simplifică construirea unei interfețe utilizator bazate pe SWT.

**PhpMyAdmin**

phpMyAdmin este un sistem de gestiune a bazelor de date MySQL liber, open source, scris în PHP și destinat administrării bazelor de date prin intermediul unui browser web. Prin phpMyAdmin se pot întreprinde diverse operații cum ar fi crearea, modificarea sau ștergerea bazelor de date, tabelelor, câmpurilor sau rândurilor; executarea de comenzi (interogări) SQL.

**MySQL**

MySQL este un sistem de gestiune a bazelor de date relaționale, produs de compania suedeza MySQL AB și distribuit sub Licența Publică Generală GNU. Este cel mai popular SGBD open-source la ora actuală, fiind o componentă cheie a stivei LAMP (Linux, Apache, MySQL, PHP).

Deși este folosit foarte des împreună cu limbajul de programare PHP, cu MySQL se pot construi aplicații în orice limbaj major. Există multe scheme API disponibile pentru MySQL ce permit scrierea aplicațiilor în numeroase limbaje de programare pentru accesarea bazelor de date MySQL, cum ar fi: C, C++, C#, Java, Perl, PHP, Python, FreeBasic, etc., fiecare dintre acestea folosind un tip specific API.

**StarUML**

StarUML este un instrument de modelare software open source care acceptă cadrul UML (Unified Modeling Language) pentru modelarea de sistem și software. Se bazează pe versiunea UML 1.4, oferă unsprezece tipuri diferite de diagrame și acceptă notația UML 2.0. Susține activ abordarea MDA (Model Driven Architecture) prin susținerea conceptului de profil UML și permițând generarea codului pentru mai multe limbi.

StarUML este construit ca un instrument modular și deschis. Oferă cadre pentru extinderea funcționalității instrumentului. Este proiectat pentru a permite accesul la toate funcțiile modelului / meta-modelului și instrumentului prin COM Automation și oferă extinderea elementelor de meniu și opțiuni. De asemenea, utilizatorii își pot crea propriile abordări și cadre conform metodologiilor lor. Instrumentul poate fi integrat și cu orice instrumente externe.

StarUML acceptă următoarele tipuri de diagrame

* Utilizați diagrama de caz
* Diagrama clasei
* Diagrama secvenței
* Diagrama de colaborare
* Diagrama graficului de stat
* Diagrama de activitate
* Diagrama componentelor
* Diagrama de implementare
* Diagrama structurii compozite

Tehnologii și frameworkuri:

**Maven** este un instrument de automatizare pentru construcții utilizat în principal pentru proiectele Java . Maven poate fi folosit și pentru a construi și gestiona proiecte scrise în C # , Ruby , Scala și alte limbi.

Maven este construit folosind o arhitectură bazată pe pluginuri care îi permite să utilizeze orice aplicație controlabilă prin intermediul intrării standard. Există și se menține un plugin pentru cadrul .NET, și un plugin nativ C / C ++ este menținut pentru Maven 2.

Tehnologii alternative precum Gradle și sbt ca instrumente de construire nu se bazează pe XML , dar păstrează conceptele cheie introduse de Maven.

Proiectele **Maven** sunt configurate folosind un model de obiect de proiect , care este stocat într-un pom.xmlfișier. Un exemplu de fișier (Figura 1) :

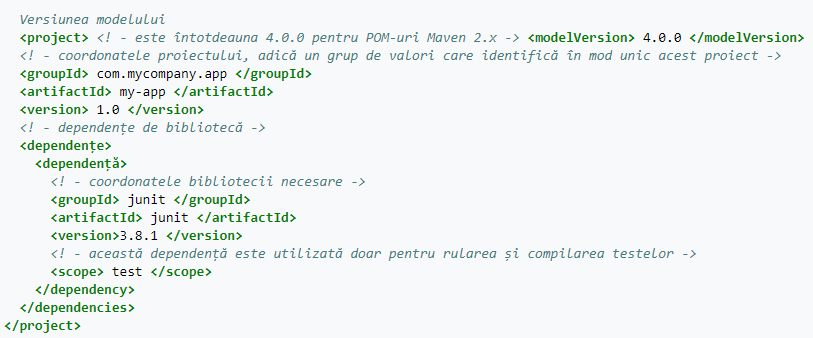


Figura 1

Acest POM definește doar un identificator unic pentru proiect ( coordonate ) și dependența acestuia de cadrul JUnit .

Cu toate acestea, acest lucru este deja suficient pentru construirea proiectului și derularea testelor unitare asociate cu proiectul. Maven realizează acest lucru prin îmbrățișarea ideii Convenției peste configurare , adică Maven furnizează valori implicite pentru configurația proiectului.

Comunicarea între client și server:

**Socket**

Socket-urile sunt o facilitate generalizata a retelelor ce au fost introduse pentru prima oara la 4.1cBSD si imbunatatite ulterior pana la forma curenta in 4.2BSD. Facilitatile socket-urilor sunt prezente in majoritatea sistemelor UNIX folosite in mod curent. (Transport Layer Interface (TLI) este alternativa System V). Socket-urile permit comunicatia dintre doua procese diferite de pe acelasi calculator sau de pe calculatoare diferite. Protocoalele pe internet sunt folosite in mod curent pentru comunicatia dintre calculatoare; alte protocoale precum DECnet pot fi folosite daca sunt prezente.

Pentru un programator un socket arata si lucreaza mai mult ca un descriptor de nivel scazut. Acest lucru se intampla deoarece comenzi ca read() si write() lucreaza cu socket-uri la fel cum lucreaza cu fisiere si cu pipe-uri. Diferentele dintre socket-uri si descriptori normali de fisiere apar la momentul crearii unui socket si la folosirea unor anumite operatii de control al unui socket. Aceste operatii sunt diferite la socket-rui fata de descriptorii normali de fisiere datorita complexitatii sporite la stabilirea unei conexiuni de retea atunci cand facem o comparatie cu accesul normal la disc.

Pentru cele mai multe operatii folosind socket-uri, rolurile de client si server trebuiesc atribuite. Un server este un proces care executa niste functii la cererea unui client.

1. **Etapa de analiză**

Unified Modeling Language (prescurtat **UML**) este un limbaj standard pentru descrierea de modele și specificații pentru software. Limbajul a fost creat de către consorțiul Object Management Group (OMG) care a mai produs printre altele și standardul de schimb de mesaje intre sisteme CORBA.

UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora (numite și obiecte). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT.

Prima versiune de UML, UML 1.0, a apărut în anul 1990 ca reacție a numeroaselor limbaje de modelare propuse pe piață. UML îi are ca fondatori pe Grady Booch, Ivar Jacobson și James Rumbaugh, așa numiții „cei trei Amigos”. Ei au dezvoltat limbajul bazându-se inclusiv pe limbaje de modelare deja existente, însă incomplete ca gamă de funcționalități.

UML oferă o largă gamă de diagrame pentru modelarea diferitelor situații în cadrul unui proiect de dezvoltare software.

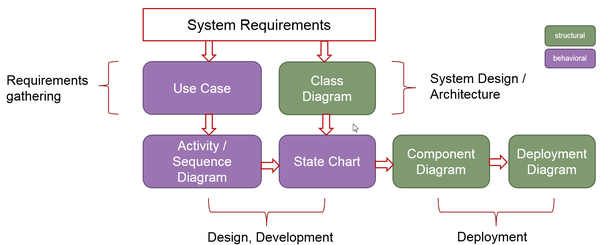


Figura 2

* 1. **Diagrama cazurilor de utilizare**

O **diagramă a cazurilor de utilizare** este cea mai simplă reprezentare a interacțiunii utilizatorului cu sistemul care arată relația dintre utilizator și diferitele *cazuri de utilizare* în care este implicat. O diagrama de cazuri de utilizare poate identifica diferitele tipuri de utilizatori ai unui sistem și diferitele cazuri de utilizare și va fi însoțită adesea și de alte tipuri de diagrame. Cazurile de utilizare sunt reprezentate fie prin cercuri, fie prin elipsă.

Scopul diagramelor de caz de utilizare este pur și simplu să ofere o vedere la nivel înalt a sistemului și să transmită cerințele în termenii persoanelor interesate . Diagrame și documentații suplimentare pot fi utilizate pentru a oferi o vedere completă funcțională și tehnică a sistemului.

Datorită naturii lor simpliste, diagramele de cazuri de utilizare pot fi un bun instrument de comunicare pentru părțile interesate . Desenele încearcă să imite lumea reală și oferă o viziune pentru părțile interesate să înțeleagă cum va fi proiectat sistemul. Siau și Lee au efectuat cercetări pentru a stabili dacă a existat o situație valabilă pentru toate diagramele cazului de utilizare sau dacă acestea nu sunt necesare.

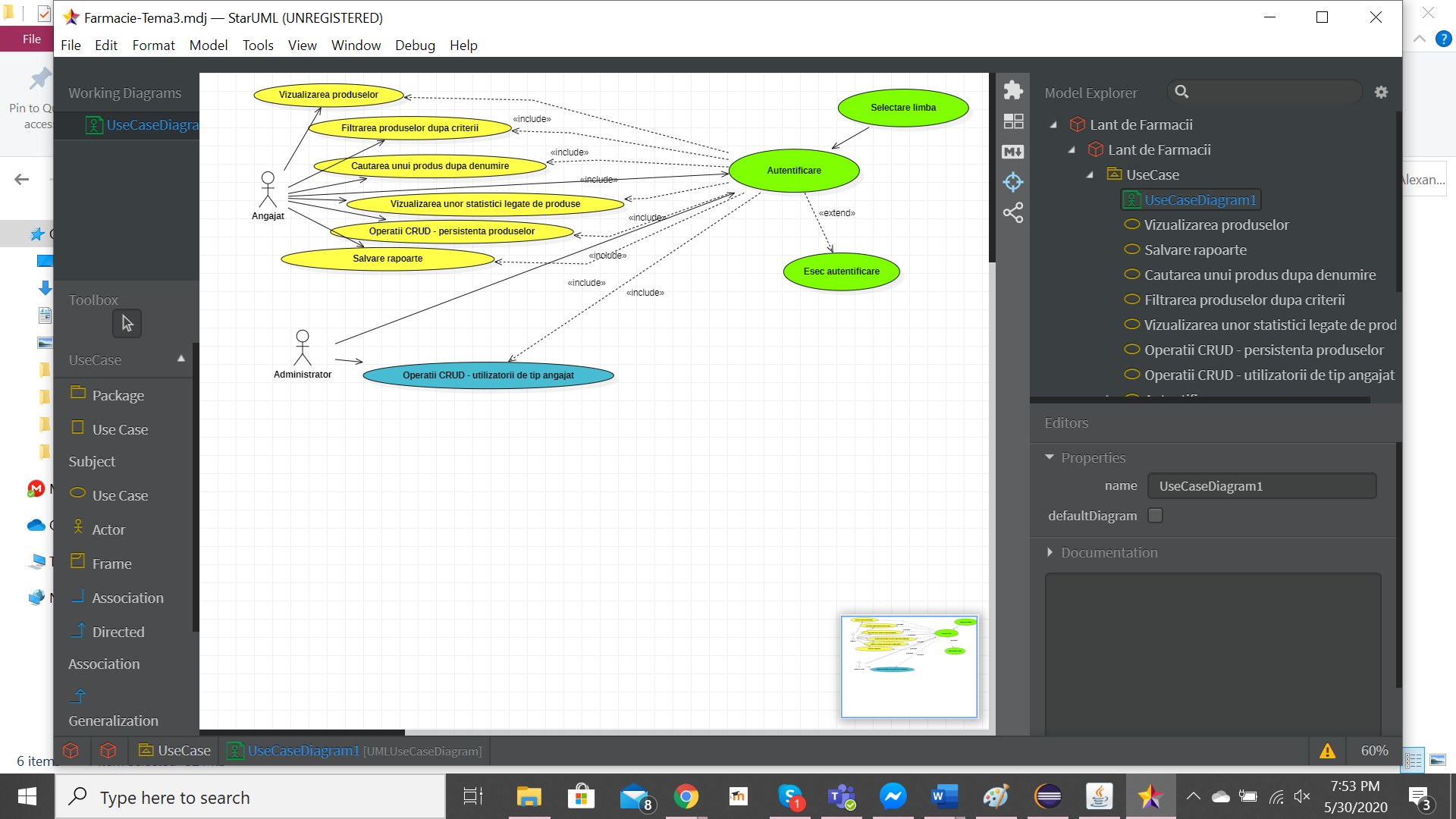


Figura 3

Figura 3 reprezintă diagrama cazurilor de utilizare pentru acest proiect. Din ea, se pot observa operațiile pe care le are fiecare utilizator de făcut, dar și necesitatea logării, sau a setării de limbă.

* 1. **Diagrama de activitate**

**Diagramele de activitate** sunt reprezentări grafice ale fluxurilor de lucru ale activităților și acțiunilor în trepte, cu suport pentru alegere, iterație și concordanță. În limbajul de modelare unificat , diagramele de activitate sunt destinate modelării atât a proceselor de calcul și de organizare (adică a fluxurilor de lucru), cât și a fluxurilor de date care se intersectează cu activitățile aferente. Deși diagramele de activitate arată în principal fluxul general al controlului, ele pot include, de asemenea, elemente care arată fluxul de date între activități prin unul sau mai multe depozite de date.

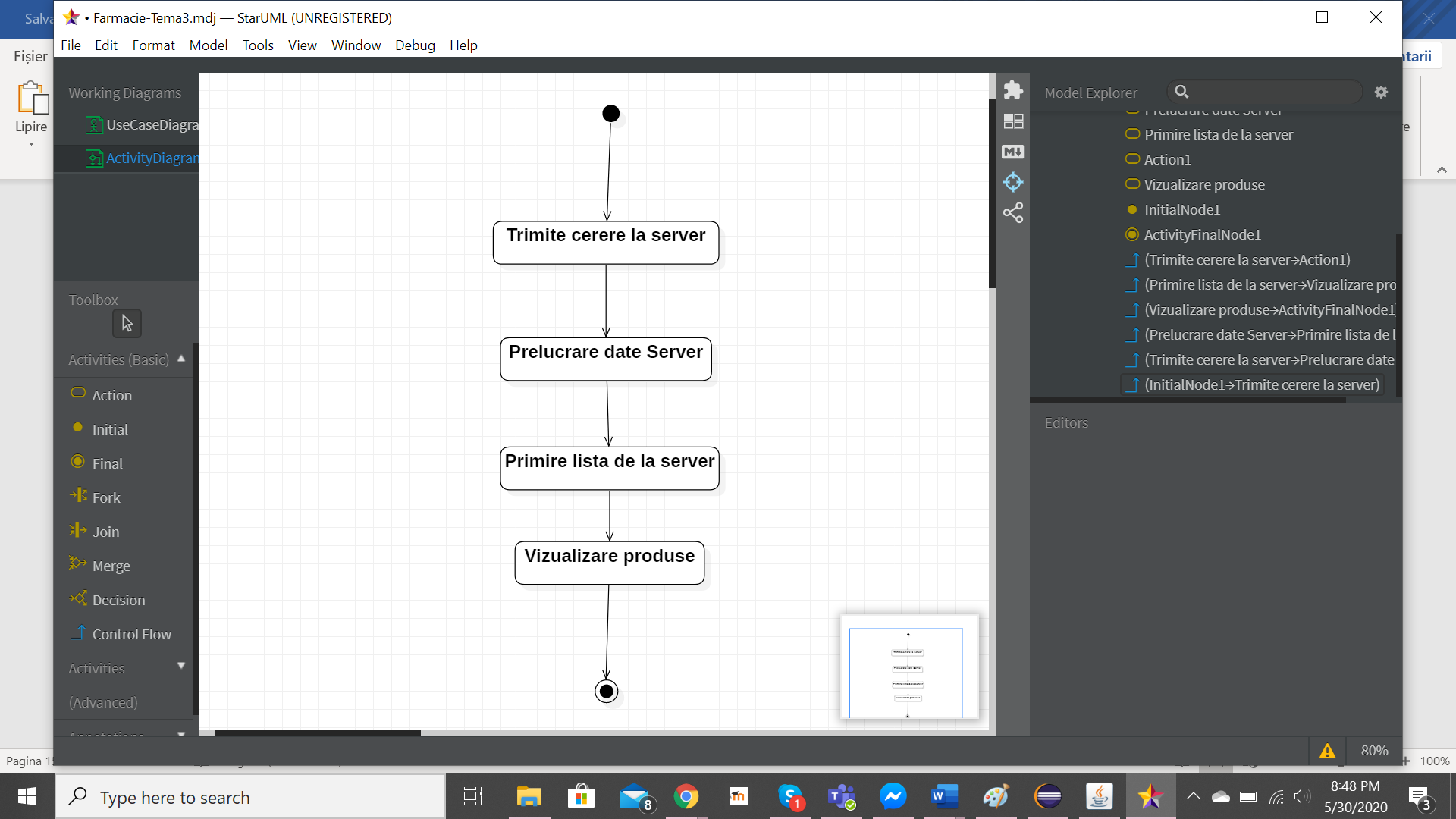
Diagramele de activitate sunt construite dintr-un număr limitat de forme, conectate cu săgeți. Cele mai importante tipuri de forme:

* *elipsele* reprezintă *acțiuni* ;
* *diamantele* reprezintă *decizii* ;
* *barele* reprezintă începutul ( *împărțirea* ) sau sfârșitul ( *unirea* ) activităților concurente;
* un *cerc negru* reprezintă începutul ( *nodul inițial* ) al fluxului de lucru;
* un *cerc negru încercuit* reprezintă sfârșitul ( *nodul final* ).

Săgețile aleargă de la început spre sfârșit și reprezintă ordinea în care se desfășoară activitățile.

**Activități angajat plafar/farmacie :**

**Vizualizare produse naturiste/medicamente:**

Figura 4

Acțiunea de a vizualiza produsele naturiste sau medicamentele se realizează după diagrama din *figura 4*:

* Se face o cerere către server, unde este stocată baza de date
* Serverul preia cererea și caută lista de produse/medicamente din baza de date
* Serverul trimite lista spre interfața grafică

* Se afiseaza produsele naturiste/medicamentele , cu ajutorul unui tabel.

**Filtrare produse naturiste/medicamente**

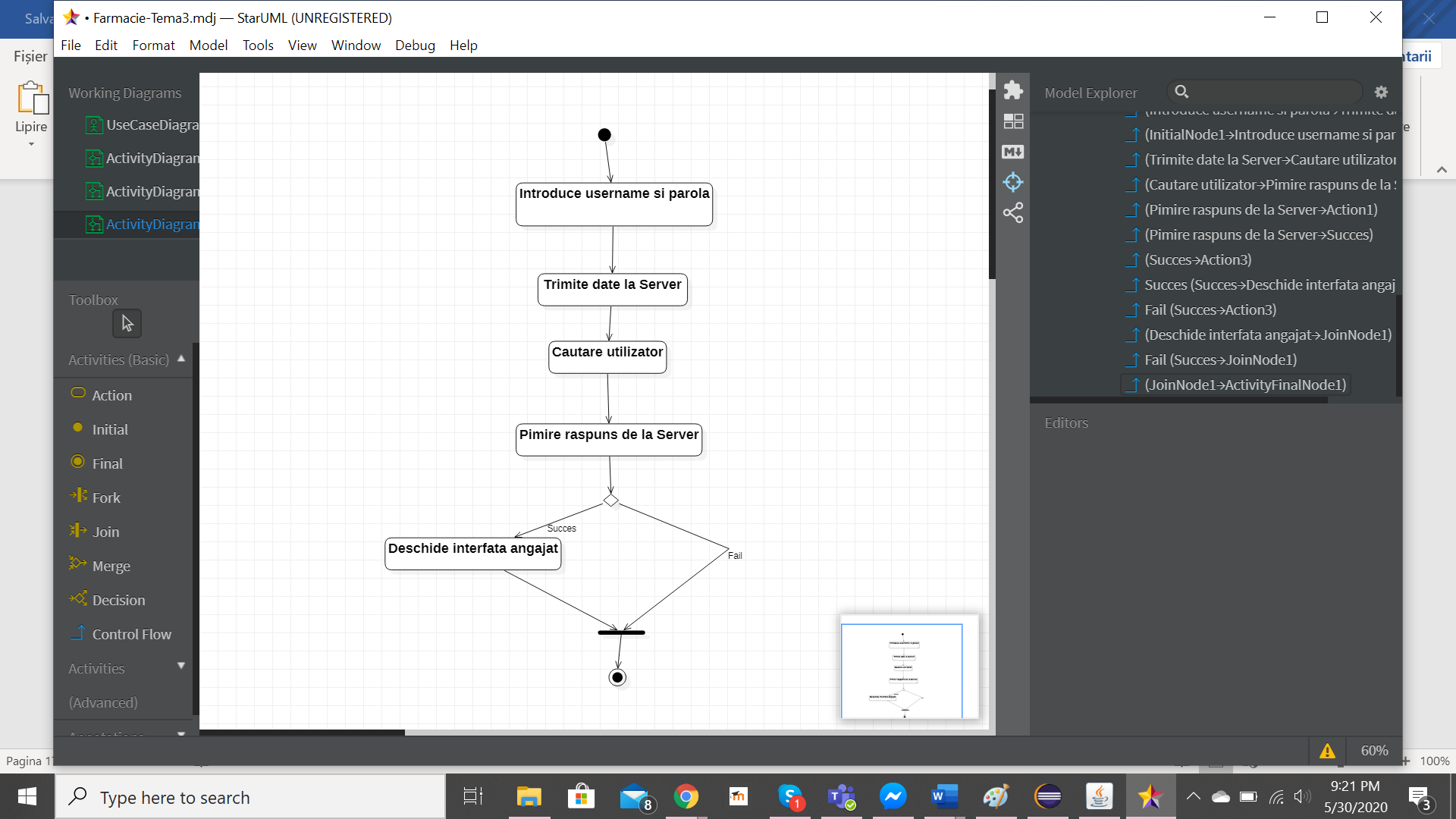
# 

# Figura 5

Acțiunea de a filtra produsele naturiste se realizează după *figura 5*:

* Se introduc datele necesare pentru filtrare
* Se trimite o cerere către server, unde este stocată baza de date
* Serverul preia cererea și caută lista de produse naturiste/medicamente din baza de date, filtrată pe baza criteriului
* Serverul trimite lista spre interfața grafică
* Se afiseaza produsele/medicamentele, cu ajutorul unui tabel.

**Logare**



# Figura 6

Acțiunea de logare pentru angajati ( plafar sau farmacie) se realizează după *figura 6* prin urmatorii pași:

* Se introduc numele și parola în casetele de text
* Se trimit datele la Server, împreună cu o cerere
* Serverul caută utilizatorul cu numele și parola respectivă
* Trimite înapoi un raspuns, dacă a gasit acel utilizator sau un mesaj de eroare

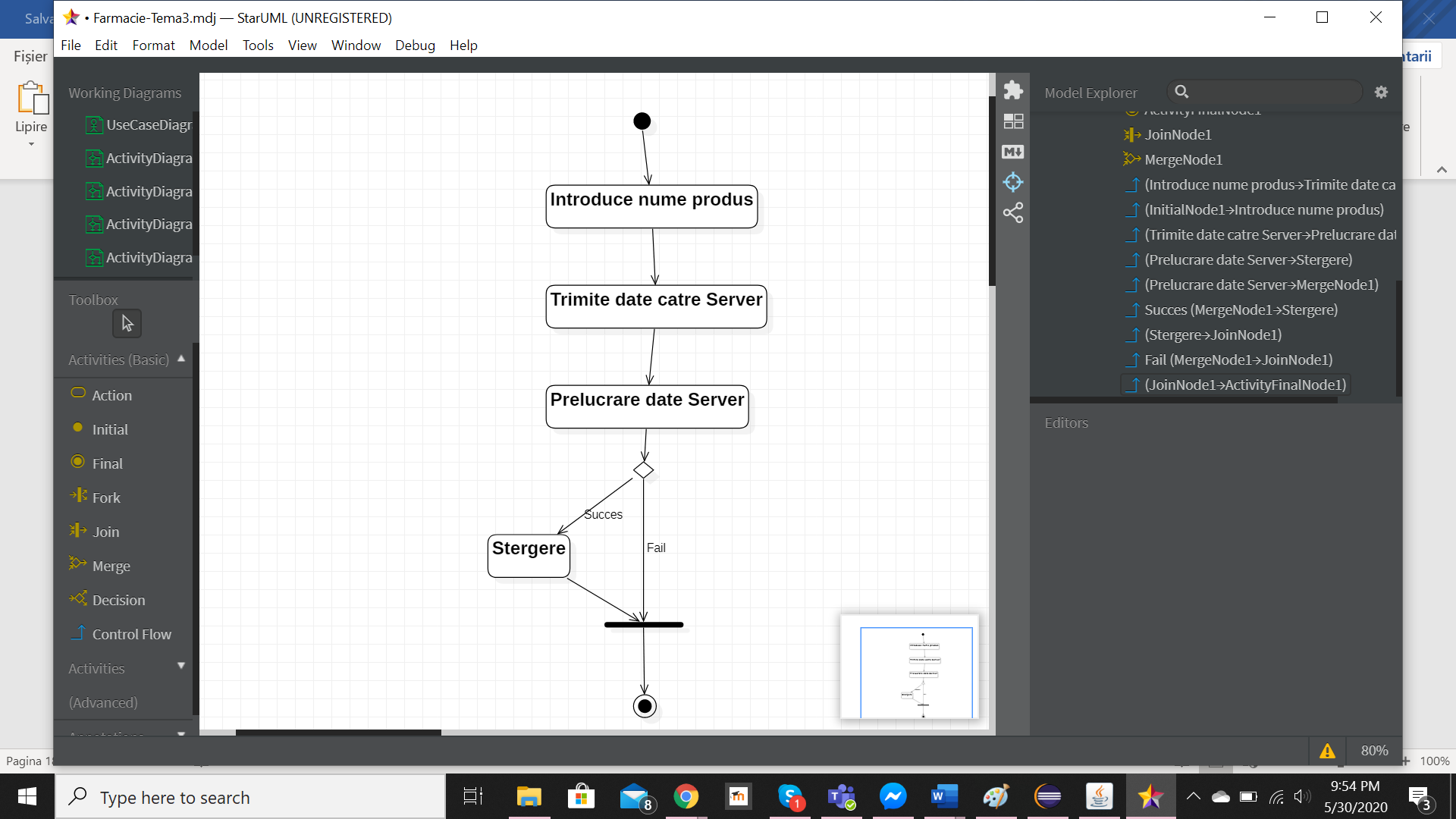
**Adăugare medicament/produs naturist**

# Figura 7

Acțiunea de a adăuga medicamente/produse naturiste se realizează după *figura 7*, prin următorii pași:

* Introduce datele necesare despre produs
* Cerere către server, unde este stocată baza de date
* Serverul preia cererea și verifică integritatea datelor
* Serverul insereaza produsul în baza de date.

**Stergere medicament/produs naturist**



Acțiunea de ștergere medicament/produs naturist se realizează după *figura 8 :*

* Se introduce numele produsului
* Se trimit datele la Server, împreună cu o cerere
* Serverul caută în baza de date produsul respectiv
* Serverul sterge produsul

Figura 8

**Modificare cantitate produs naturist/ medicament**

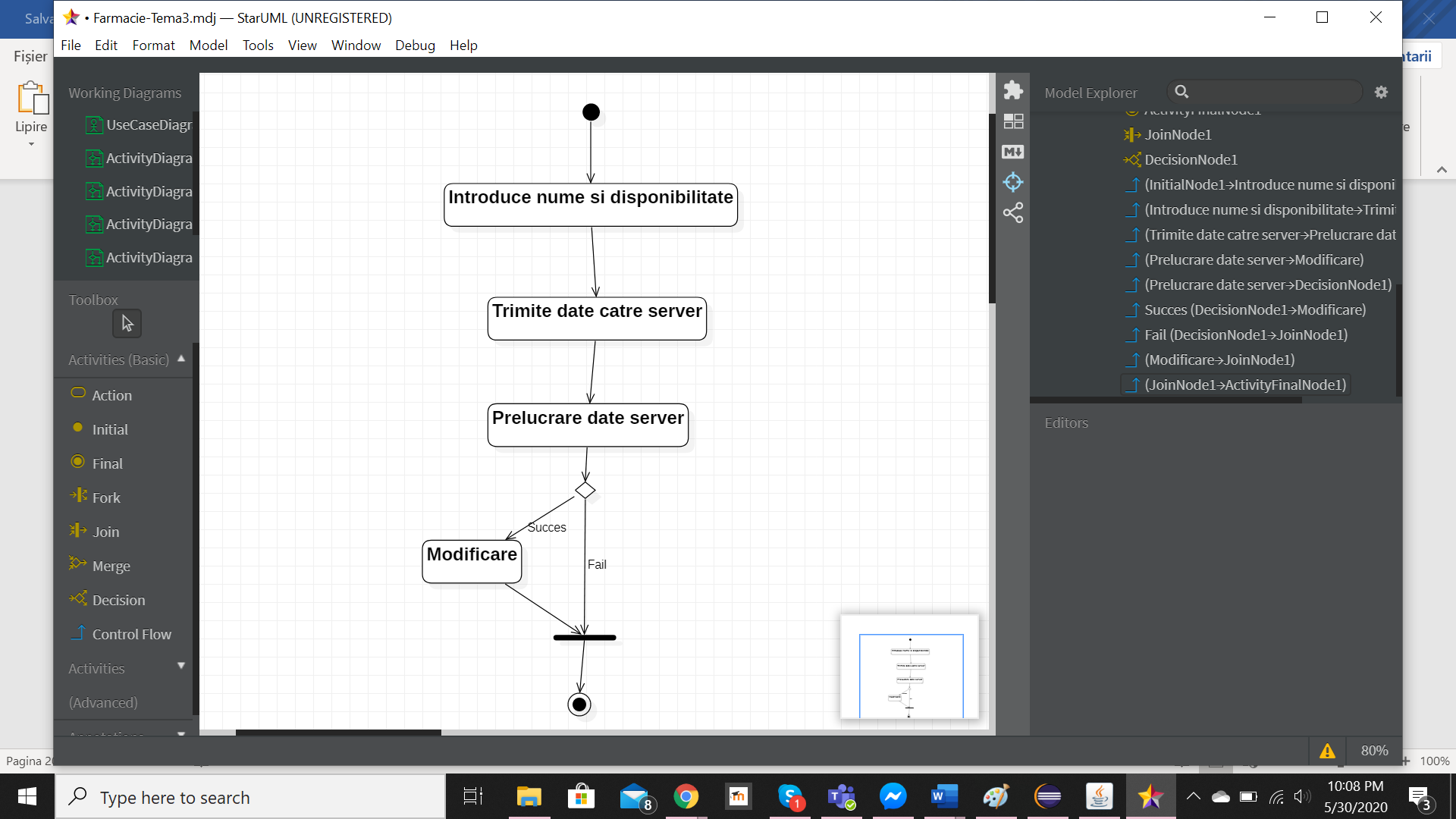


Figura 9

Acțiunea de modificare a cantitatii produselor naturiste/medicamentelor se realizează după *figura 9* :

* Se introduc datele necesare realizării acțiunii
* Se trimit datele la Server, împreună cu o cerere
* Serverul caută în baza de date produsul respectiv
* Serverul modifica produsul

**Activități administrator**

**Adăugare angajat**



Figura 10

Acțiunea de a adăuga angajat se realizează după *figura 10*, prin următorii pași:

* Introducere date despre angajat
* Cerere către server, unde este stocată baza de date
* Serverul preia cererea și verifică integritatea datelor
* Serverul insereaza noul angajatul în baza de date

**Vizualizare angajați**

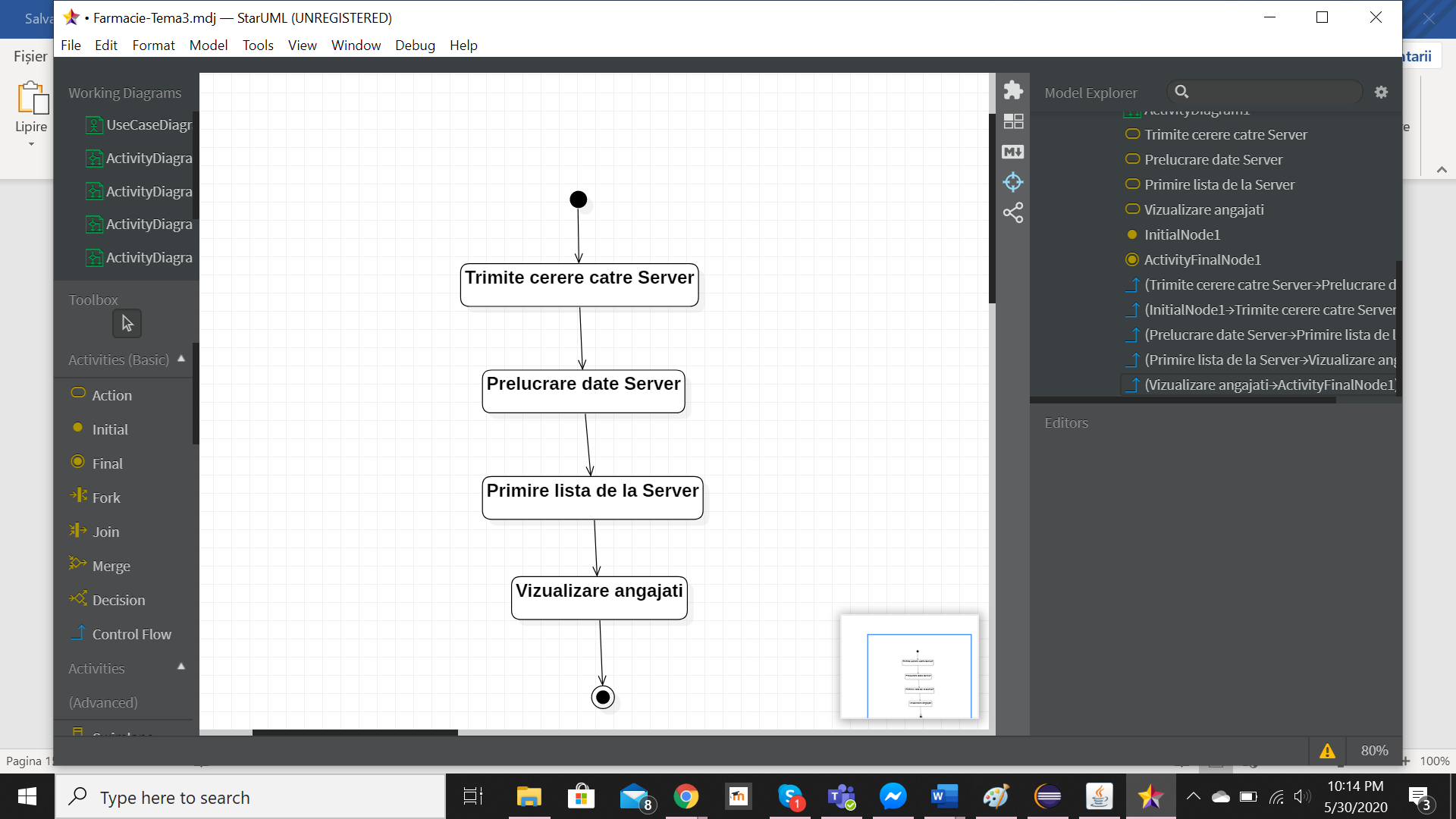
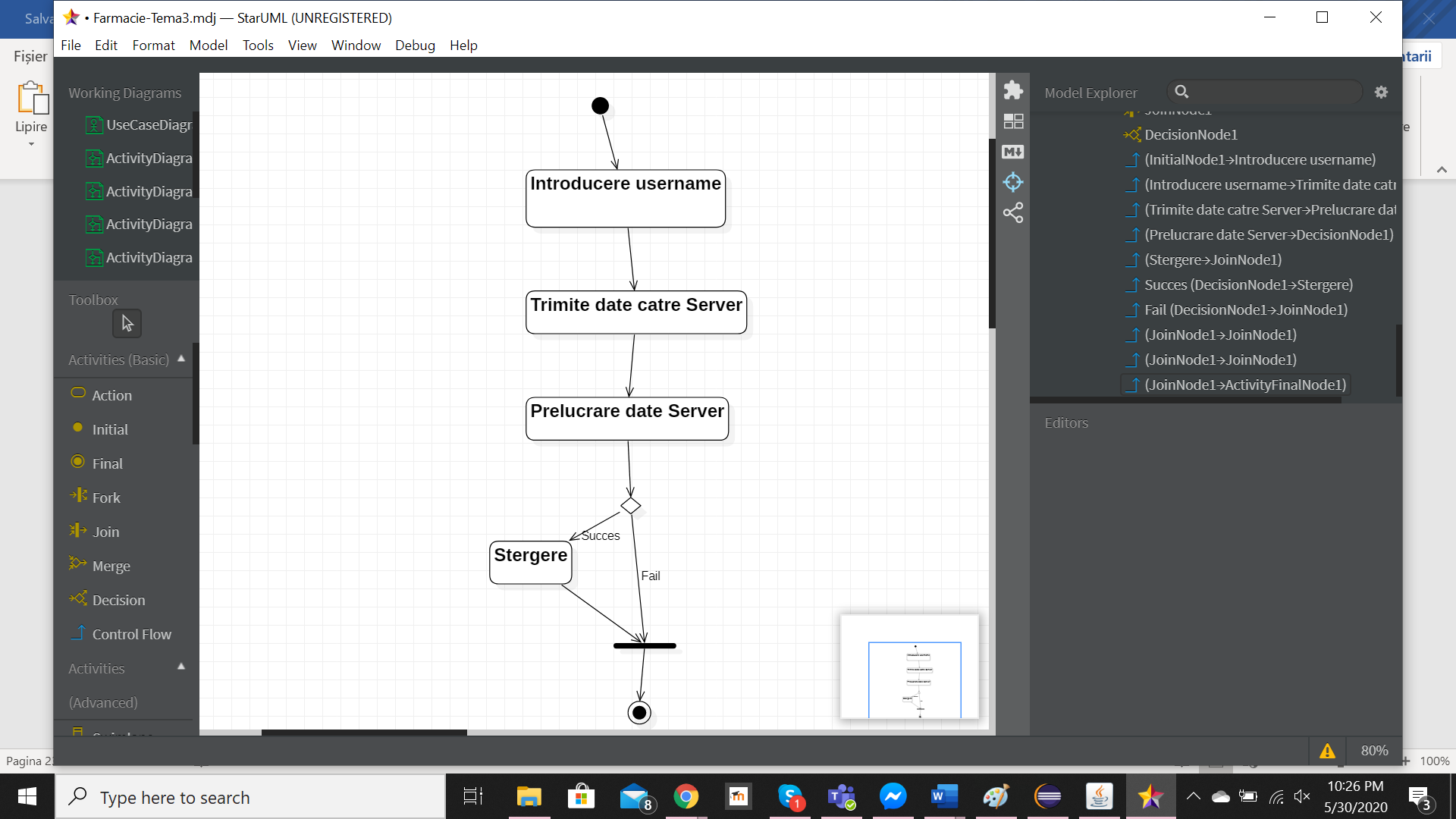


Figura 11

Acțiunea de a vizualiza angajați se realizează după *figura 11*, prin următorii pași:

* Cerere către server, unde este stocată baza de date
* Serverul preia cererea și caută lista de angajați din baza de date
* Serverul trimite lista spre interfața grafică
* Se afiseaza angajații, cu ajutorul unui tabel.

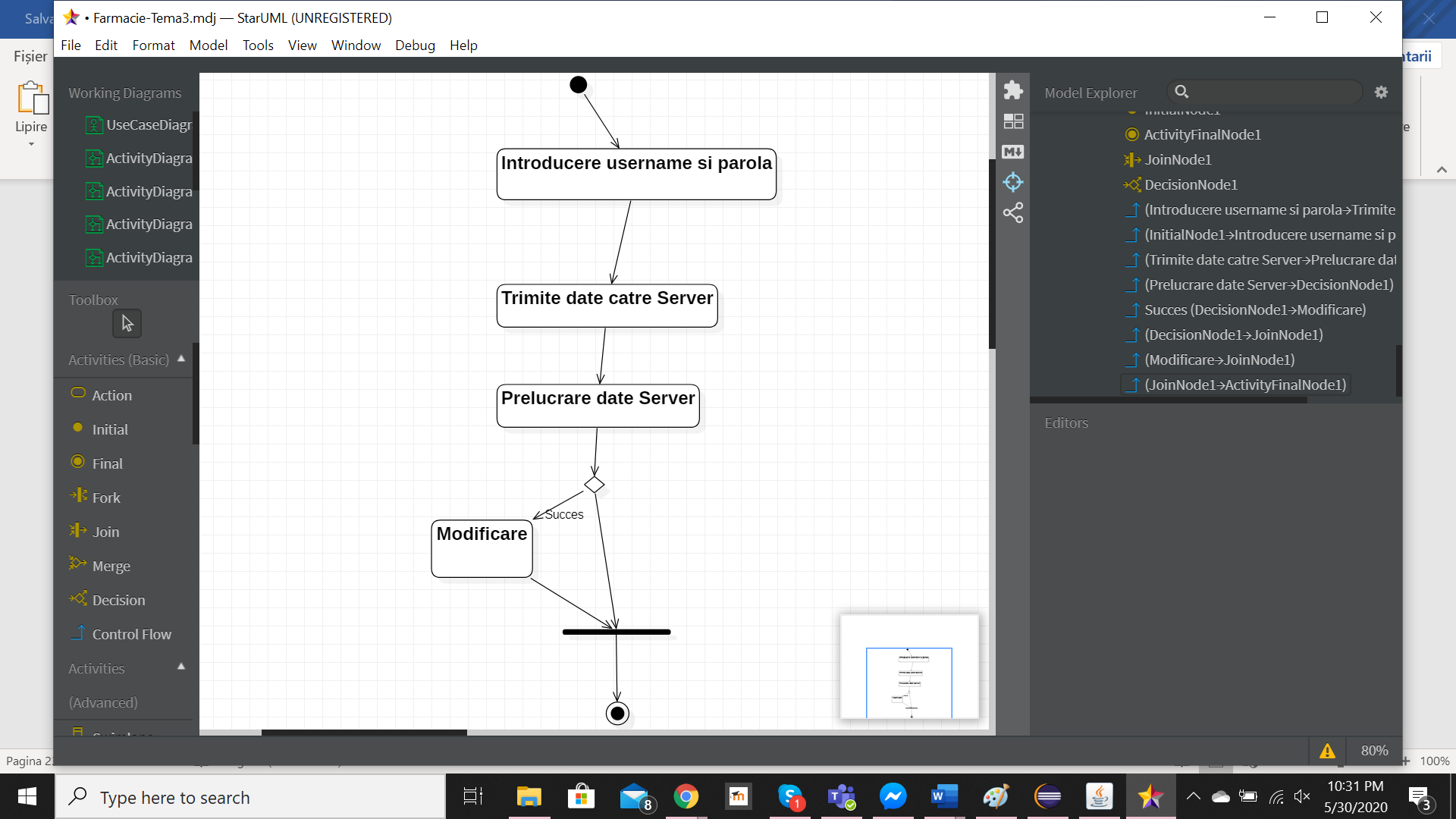
**Stergere angajat**

 Figura 12

Acțiunea de ștergere angajat se realizează după *figura 12* :

* Se introduce numele de utilizator
* Se trimit datele la Server, împreună cu o cerere
* Serverul caută în baza de date angajatul respectiv
* Serverul sterge angajatul respectiv

**Modificare parola angajat**

 Figura 13

Acțiunea modificare a parolei unui angajat se realizează după *figura 13*:

* Se introduc datele necesare realizării acțiunii
* Se trimit datele la Server, împreună cu o cere
* Serverul caută în baza de date angajatul respectiv
* Serverul modifica parola angajatului

1. **Etapa de proiectare**

Sarcina acestei faze este de a detalia modelul obiect rezultat din faza de analiza. Se va determina sarcina exacta a obiectelor si structurile de date respectiv algoritmii necesari pentru faza de implementare. Tot in aceasta faza se construiesc diagrame de clase si de obiecte precum si  diagrame de colaborare intre obiecte.

* 1. **Diagrama de clase**

În inginerie software , o **diagramă de clasă** în limbajul unificat de modelare (UML) este un tip de diagramă de structură statică care descrie structura unui sistem, arătând clasele sistemului , atributele, operațiile (sau metodele) și relațiile dintre obiecte.

Diagrama de clase este blocul principal de modelare orientat pe obiecte . Este utilizat pentru modelarea conceptuală generală a structurii aplicației și pentru modelarea detaliată a traducerii modelelor în cod de programare . Diagramele de clasă pot fi de asemenea utilizate pentru modelarea datelor .  Clasele dintr-o diagramă de clase reprezintă atât elementele principale, interacțiunile din aplicație, cât și clasele care trebuie programate.

În diagrama, clasele sunt reprezentate cu cutii care conțin trei compartimente:

* Compartimentul de sus conține numele clasei. Este tipărită cu caractere aldine și centrată, iar prima literă este scrisă cu majuscule.
* Compartimentul din mijloc conține atributele clasei. Sunt aliniate la stânga, iar prima literă este cu litere mici.
* Compartimentul de jos conține operațiunile pe care le poate executa clasa. De asemenea, sunt aliniate la stânga și prima literă este cu minuscule.
  + 1. **Șablon arhitectural Client-Server**

Modelul **client-server** este o structură de aplicație distribuită care distribuie sarcini sau sarcini de muncă între furnizorii unei resurse sau servicii, numiți servere și solicitanți de servicii, numiți clienți. Adesea, clienții și serverele comunică printr-o rețea de calculatoare pe hardware separat, dar atât clientul cât și serverul pot sta în același sistem. O gazdă server rulează unul sau mai multe programe server, care își împărtășesc resursele cu clienții. Un client nu împărtășește nici una dintre resursele sale, dar solicită conținut sau serviciu de la un server. Prin urmare, clienții inițiază sesiuni de comunicare cu servere, care așteaptă solicitări primite.

Caracteristica **client-server** descrie relația programelor care colaborează într-o aplicație. Componenta server oferă o funcție sau un serviciu unuia sau mai multor clienți, care inițiază cereri pentru astfel de servicii. Serverele sunt clasificate în funcție de serviciile pe care le oferă. De exemplu, un server web servește pagini web și un server de fișiere servește fișiere de calculator . O resursă partajată poate fi oricare dintre componentele software și componente ale computerului serverului, de la programe și date la procesoare și dispozitive de stocare . Partajarea resurselor unui server constituie un serviciu .

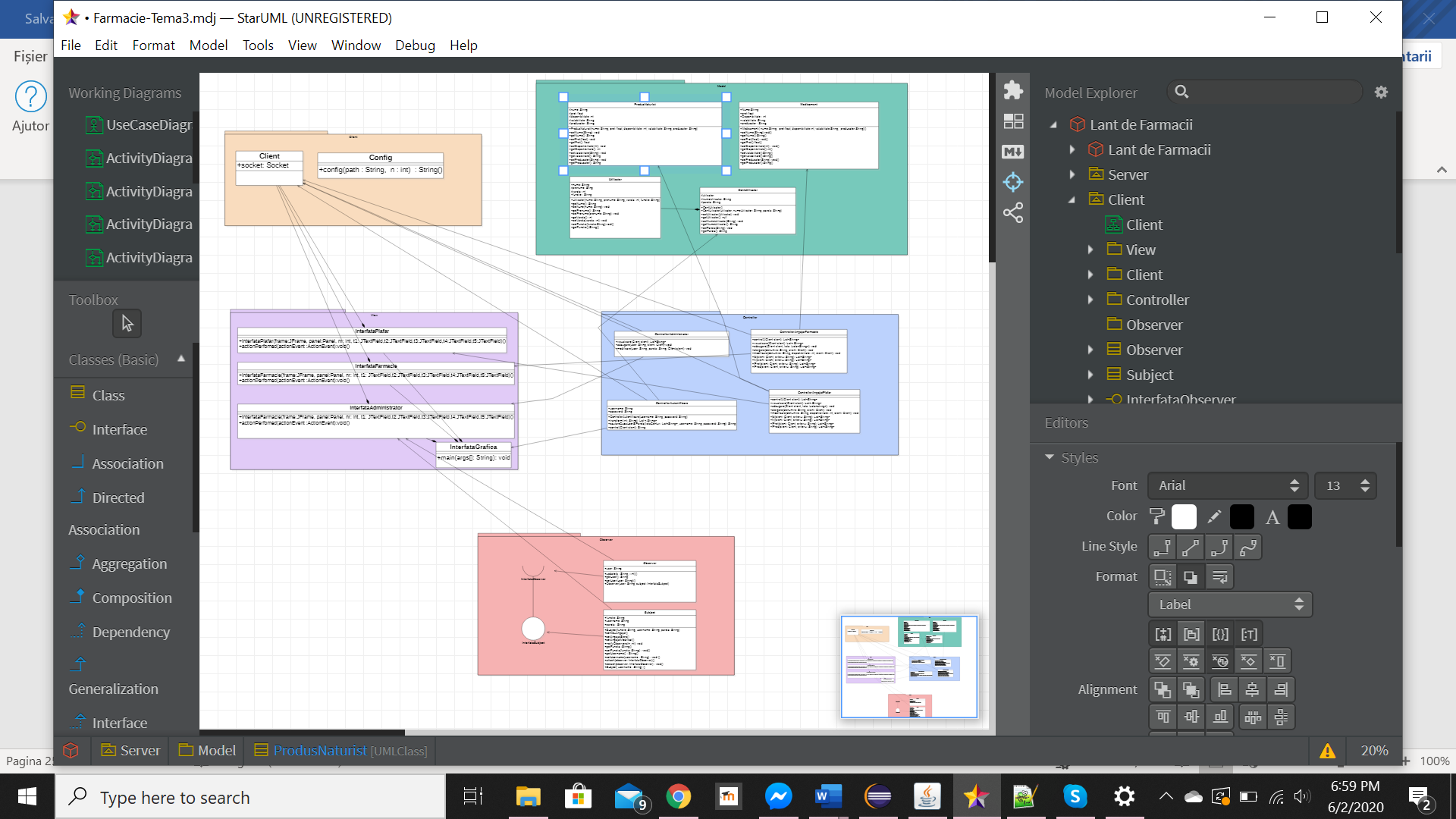


Figura 14

Diagrama reprezentată în *figura 14* reprezintă diagrama de clase pentru aplicația Client. În acestă aplicație sunt stocate părțile de afișare: interfețele grafice, partea de control al acestora și partea de conexiune cu serverul.

Pachetul de culoare mov reprezintă pachetul View care contine interfețele utilizator pentru fiecare tip de persoane: angajat plafar, angajat farmacie și administrator. Fiecare din aceste clase creează obiecte de tip swing: paneluri, frame, butoane, casete de text etc. Cele 3 sunt cuprinse într-o clasa mare, fiind clasa main.

Pachetul de culoare albastru reprezintă pachetul Controller care contine controlul pentru fiecare interfață grafică, în plus am mai adăugat și controlul pentru autentificare, utilizatorii fiind nevoiți să se logheze pentru a putea folosi aplicația. Fiecare din aceste clase au rolul de a controla ce activități se întâmplă în momentul în care se apasă un anumit buton sau în momentul în care este nevoie de calcularea unor parametrii suplimentari.

Pachetul de culoare galben reprezinta pachetul Client ce contine conexiunea cu serverul. Clasa client realizeaza cererile necesare fiecăror instrucțiuni, iar clasa config are rolul de a seta o anumită configurație a clientului și serverului: să aibă amandoua același port sau aceeași adresă ip.

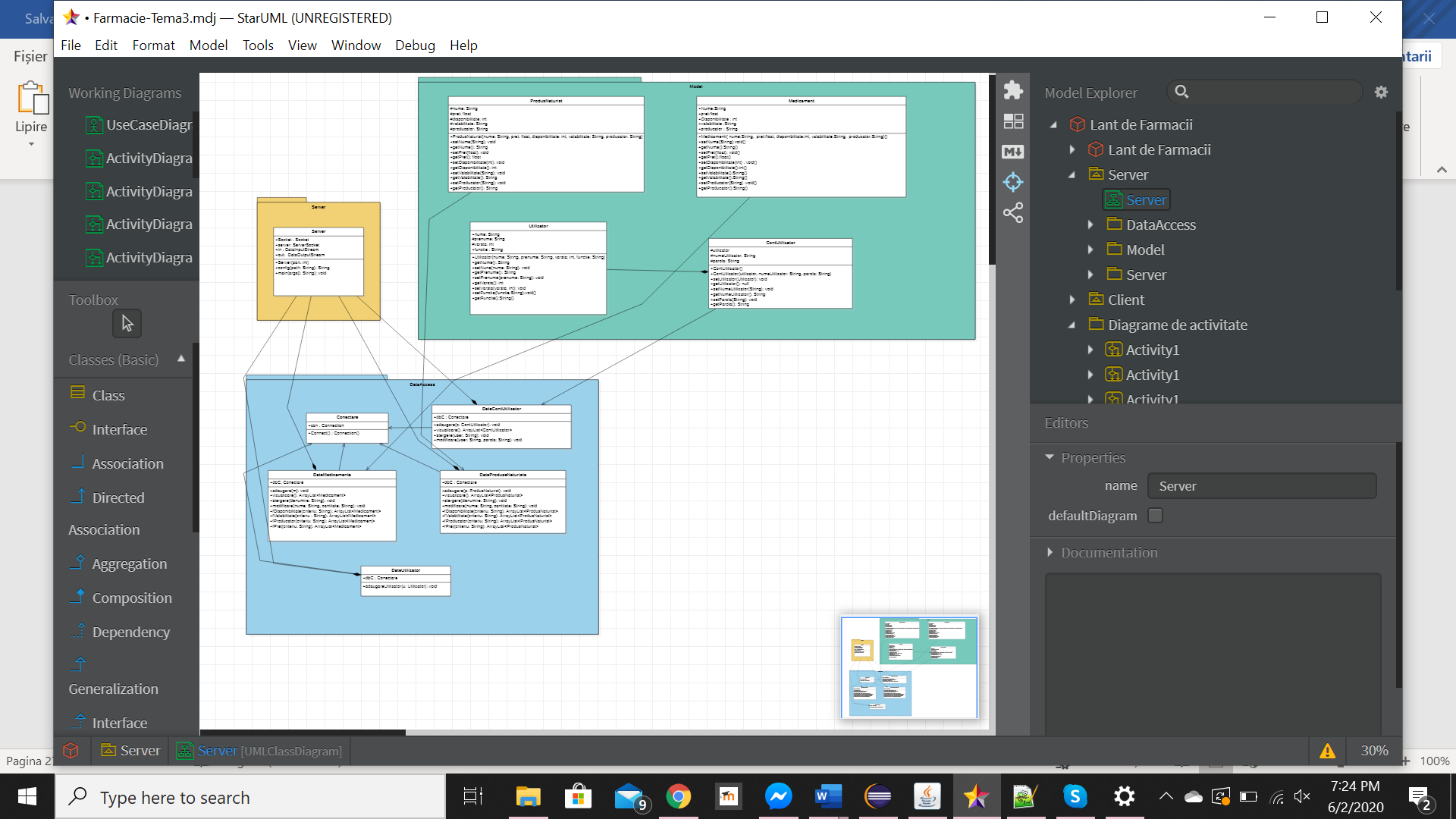


Figura 15

Diagrama reprezintă diagrama de clase pentru aplicația Server. În acestă aplicație sunt stocate părțile de model și părțile de conexiune și operații cu baza de date și conexiunea cu Clinetul.

Pachetul de culoare verde reprezintă clasele de model.

Pachetul de culoare albastru reprezintă lucrul cu baza de date. O clasă dintre acestea realizează conexiunea la baza de date. Are ca și câmp o conexiune și are doar implementat doar un constructor în corpul clasei. Acest constructor initializeaza câmpul cu conexiunea realizata de getConnection care primește ca și argumente un host, acesta fiind 3306 standard pentru localhost-ul fiecarui utilizator, un user și o parola, dacă baza de date implementata are sau nu parola. În cazul în care nu s-a putut stabili o conexiune cu baza de date se afiseaza un mesaj de eroare. Restul claselor realizează interogări în baza de date: operații de insert update sau delete, pe fiecare tabelă în parte.

Pachetul de culoare galbena reprezintă conexiunea cu clientul. Clasa Server realizeaza cererile necesare fiecăror instrucțiuni, metoda config din clasa respectivă are rolul de a seta o anumită configurație a clientului și serverului: să aibă amandoua același port sau aceeași adresă ip.

* + 1. **Șabloane de proiectare**
       1. **Observer**

Modelul de observare este un model de proiectare software în care un obiect , numit subiect , menține o listă a dependenților săi, numiți observatori și îi notifică automat cu privire la orice schimbare de stare, de obicei apelând la una dintre metodele lor .

Este utilizat în principal pentru implementarea procesării distribuite a evenimentelorsisteme, în software-ul „condus de evenimente”. În aceste sisteme, subiectul este de obicei denumit „flux de evenimente”, în timp ce observatorii sunt numiți „scufundarea evenimentelor”.

Modelul Observer abordează următoarele probleme:

* Ar trebui definită o dependență de la unu la mai mulți dintre obiecte, fără ca obiectele să fie strâns cuplate.
* Trebuie să se asigure că atunci când un obiect schimbă starea, un număr deschis de obiecte dependente este actualizat automat.
* Ar trebui să fie posibil ca un obiect să notifice un număr deschis de alte obiecte.

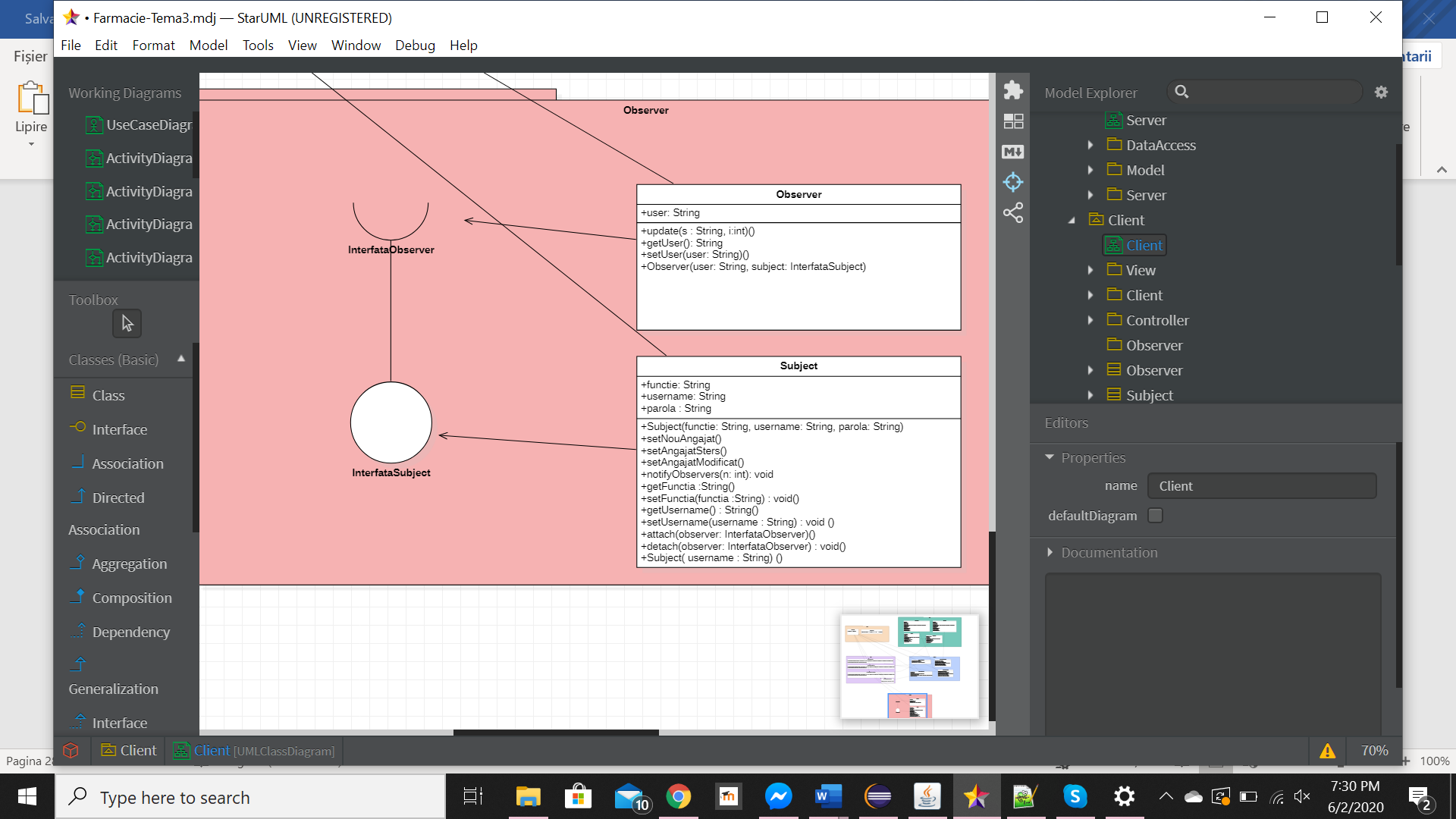


Figura 16

După cum se poate observa în *figura 16* și din diagrama de clase, operațiile necesare pentru șablonul observer sunt implementate pentru operațiile administratorului astfel că în momentul în care un adminstratoru introduce, modifică sau șterge un angajat, toți angajații sunt notificați în legătură cu acest lucru, inclusiv angajatul respectiv.

* + - 1. **Factory**

Modelul metodei de fabrică este un model creațional care folosește metode din fabrică pentru a rezolva problema creării obiectelor, fără a fi necesar să specifici clasa exactă a obiectului care va fi creat.

Acest lucru se realizează prin crearea de obiecte prin apelarea unei metode de fabrică - fie specificate într-o interfață și implementate de către clase de copii, fie implementate într-o clasă de bază și, în mod opțional, sunt suprasolicitate de clase derivate, mai degrabă decât prin apelarea unui constructor.

Modelul de proiectare a metodei de fabrică este utilizat în loc de constructorul clasei obișnuite pentru păstrarea în cadrul principiilor SOLID de programare, decuplarea construcției de obiecte de la obiectele în sine.

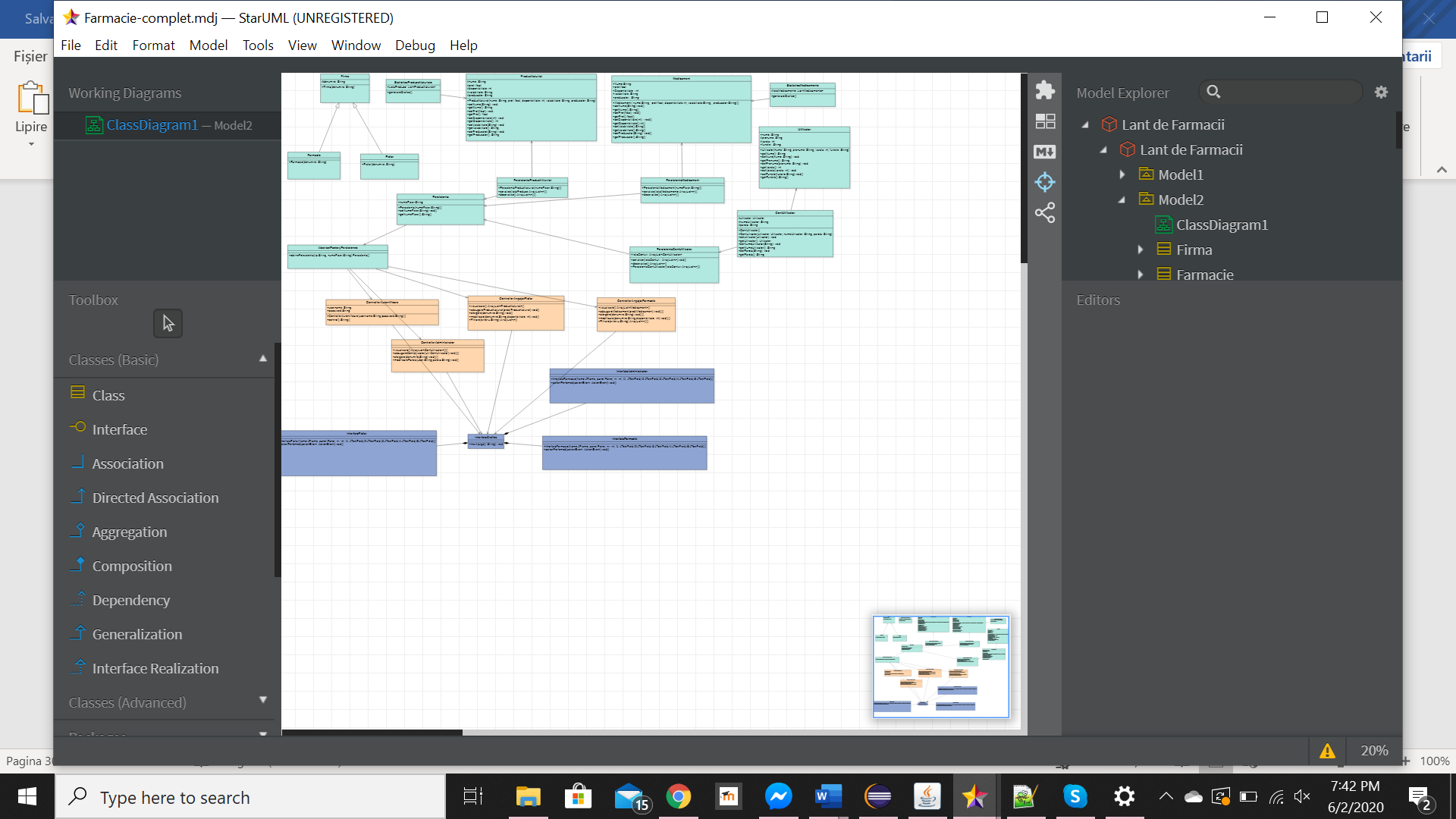


Figura 17

Schema de proiectare a metodei de fabrică este utilizată mai întâi prin definirea unei operațiuni separate, a unei metode de fabrică , pentru crearea unui obiect și apoi folosirea acestei metode din fabrică, apelând-o pentru a crea obiectul.

* + - 1. **Builder**

Modelul constructor este un model de proiectare conceput pentru a oferi o soluție flexibilă la diverse probleme de creare a obiectelor în programarea orientată pe obiecte . Intenția modelului de proiectare a constructorului este de a separa construcția unui obiect complex de reprezentarea sa.

Modelul de design Builder rezolvă probleme precum:

* Cum poate o clasă (același proces de construcție) să creeze reprezentări diferite ale unui obiect complex?
* Cum poate fi simplificată o clasă care include crearea unui obiect complex?

Crearea și asamblarea părților unui obiect complex direct în cadrul unei clase este inflexibil. Angajează clasa să creeze o reprezentare particulară a obiectului complex și face imposibilă schimbarea reprezentării ulterior independent de (fără a fi necesară schimbarea) clasei.

Modelul de design al Builder descrie cum să rezolvi astfel de probleme:

* Incapsulați crearea și asamblarea părților unui obiect complex într-un Builderobiect separat .
* O clasă delege crearea unui Builderobiect în loc de a crea direct obiectele.
  1. **Baza de date**

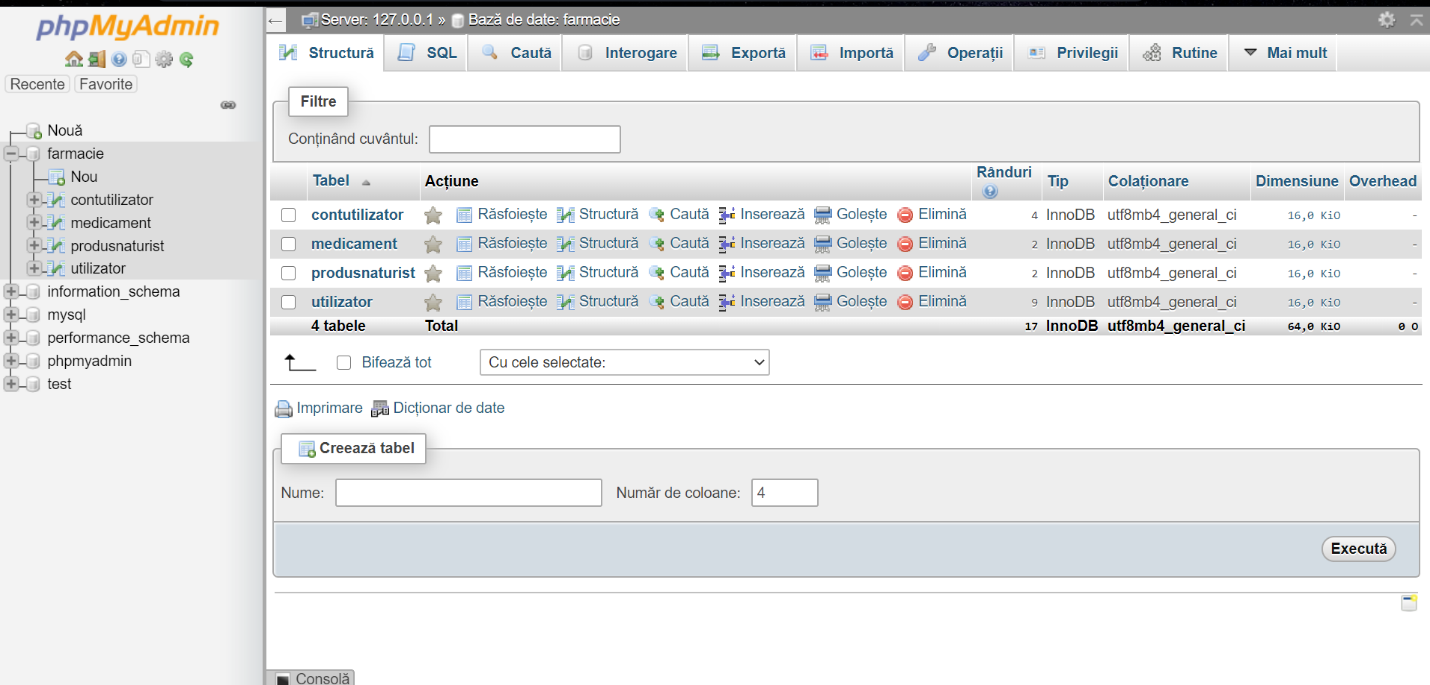


Figura 18

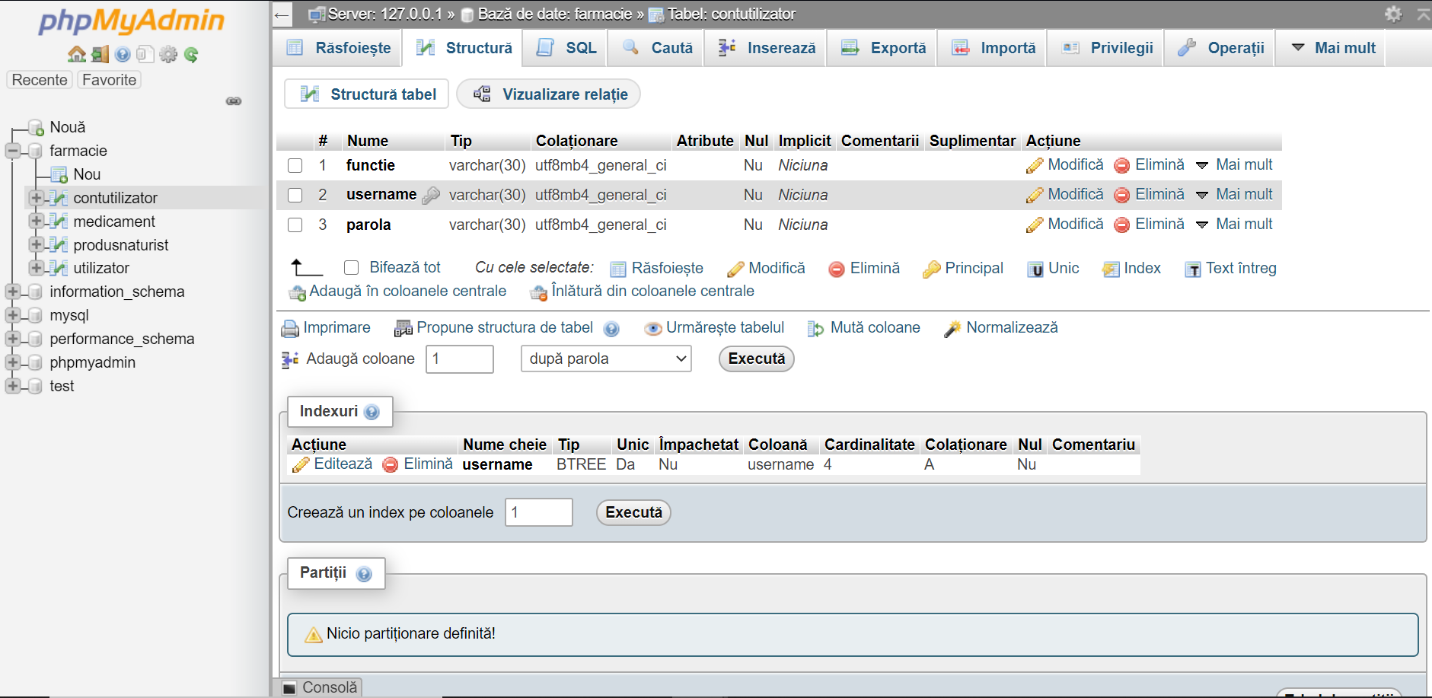
****

Figura 19 - Structura tabelului contUtilizator

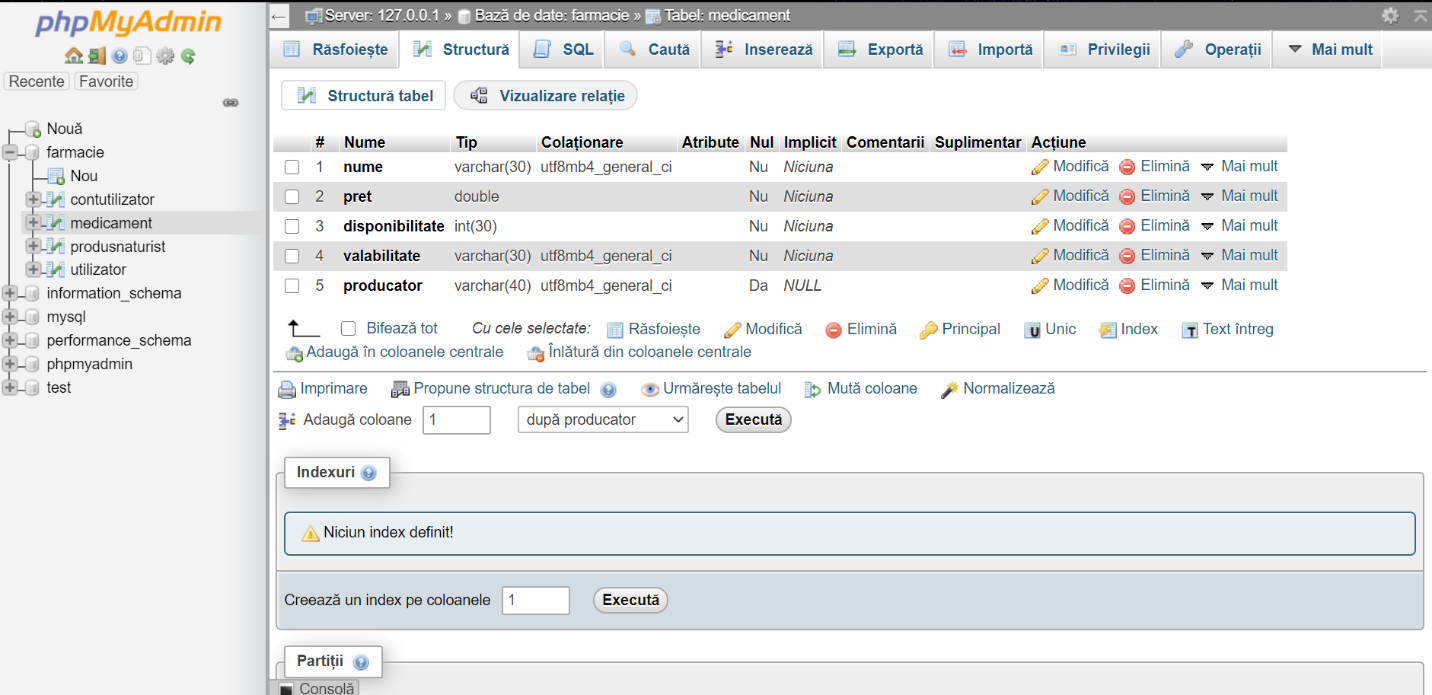


Figura 20 – Structura tabelului medicament

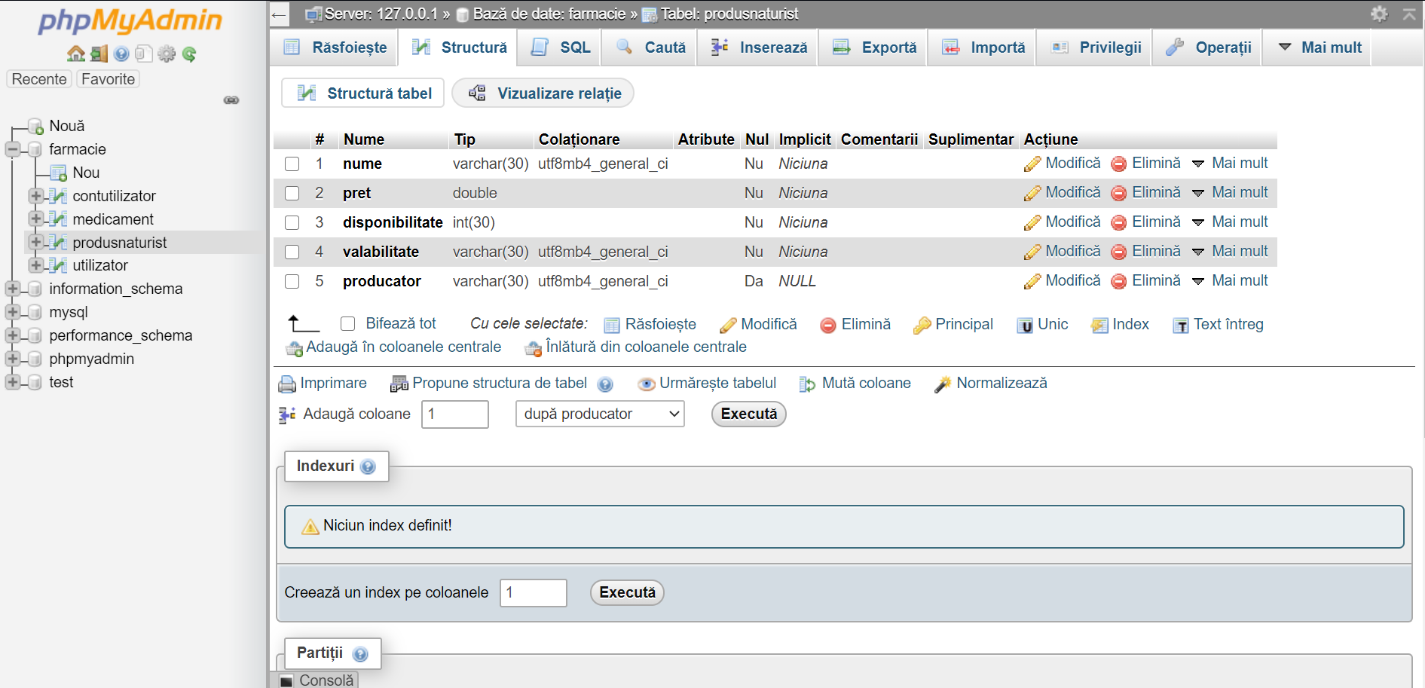


Figura 21– Structura tabelului produsNaturist

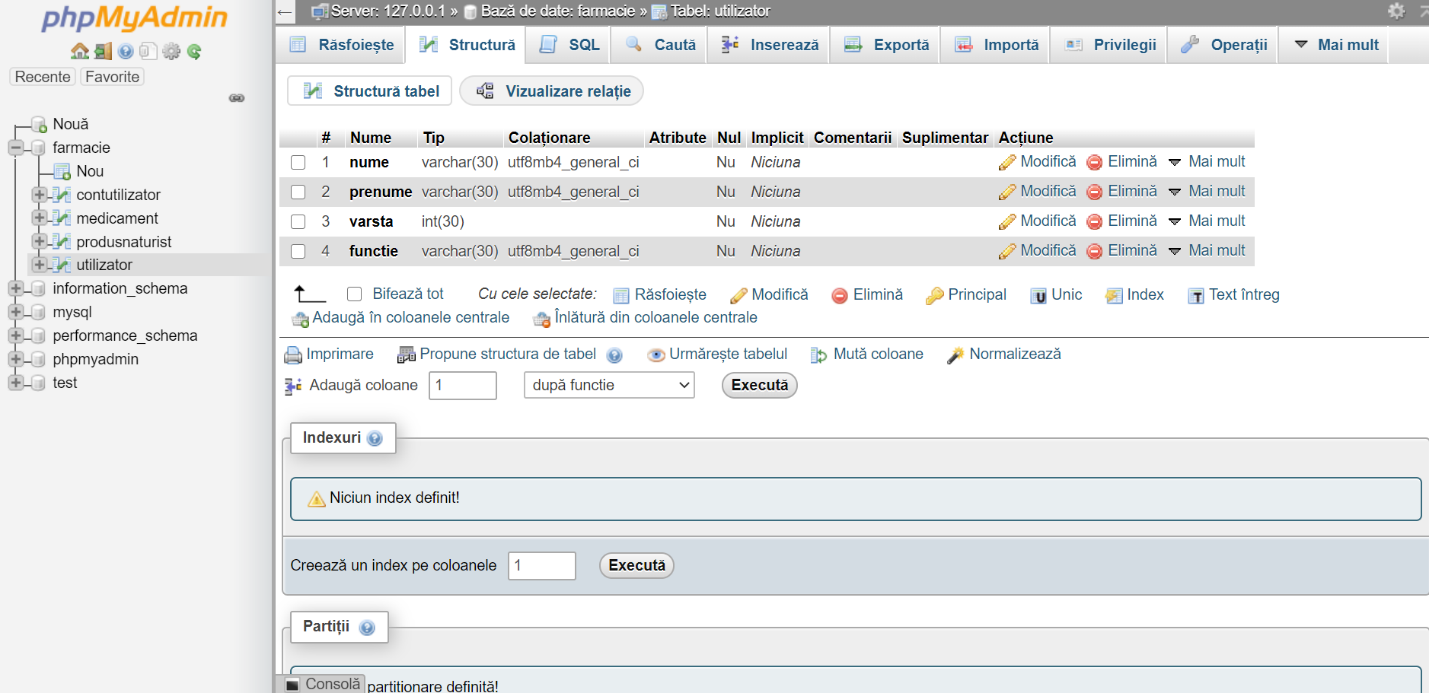


Figura 22– Structura tabelului utilizator

1. **Manual de utilizare**
2. Se pornește aplicația Server si se asteapta confirmarea de la acesta.

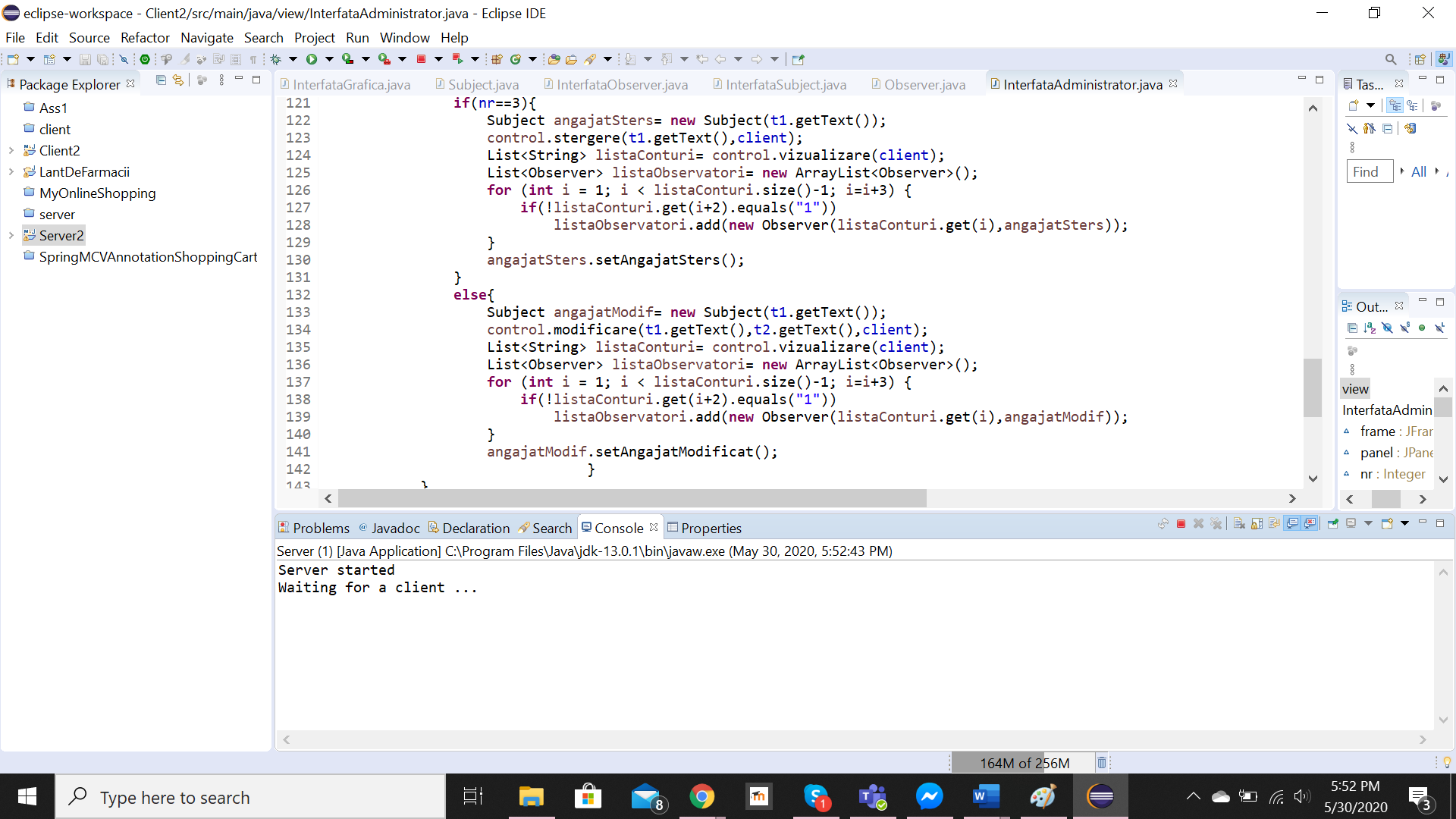


Figura 22

1. Se pornește aplicația Client. Cand porneste va apărea opținea de selectare a unei limbi: română, engleză sau franceză.
2. Se selectează limba și se apasă butonul „Selectare”.

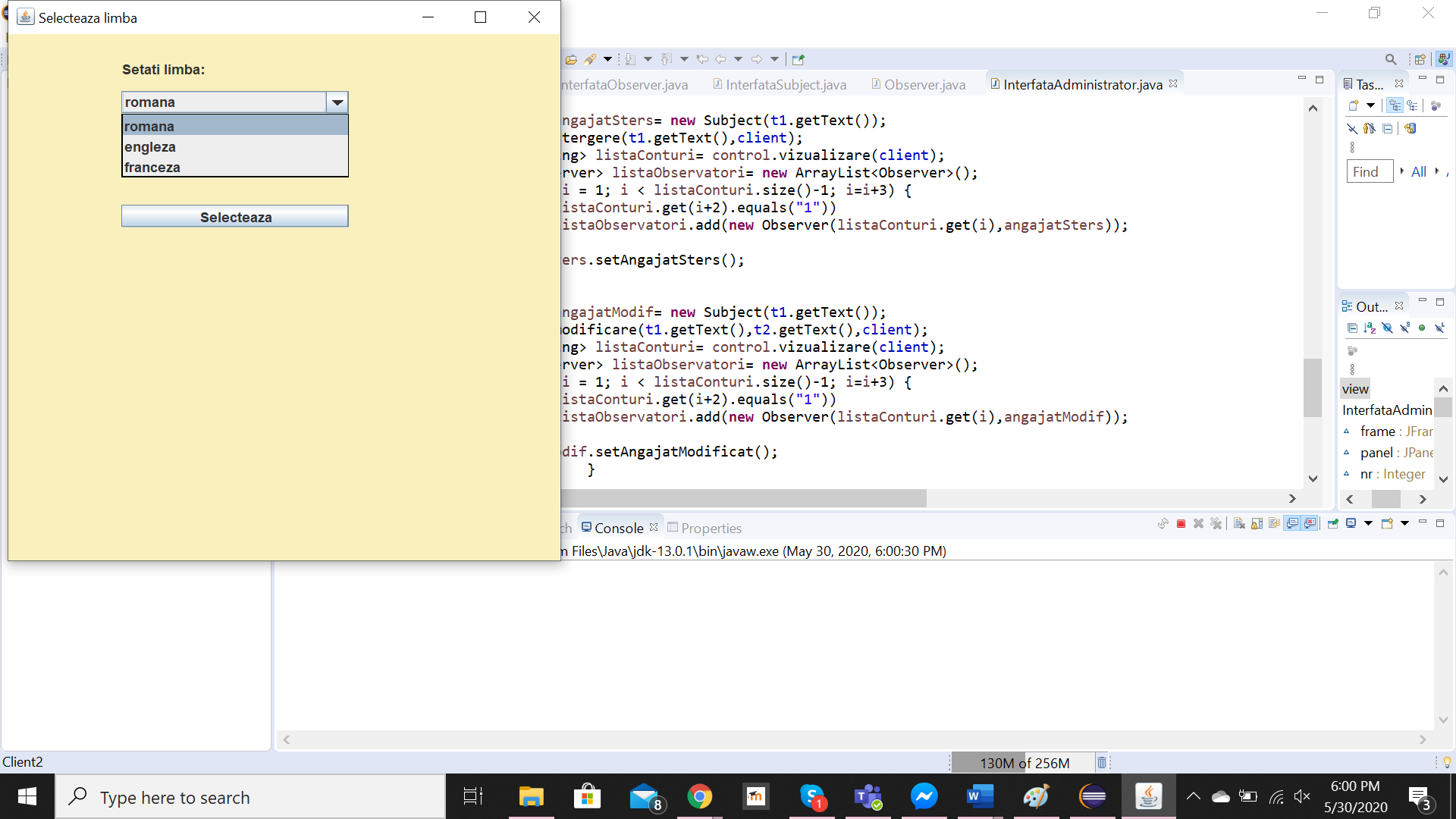


Figura 23

1. Se închide automat fereastra de setare a limbii, urmând deschiderea interfeței mari unde trebuie introduse datele de logare.

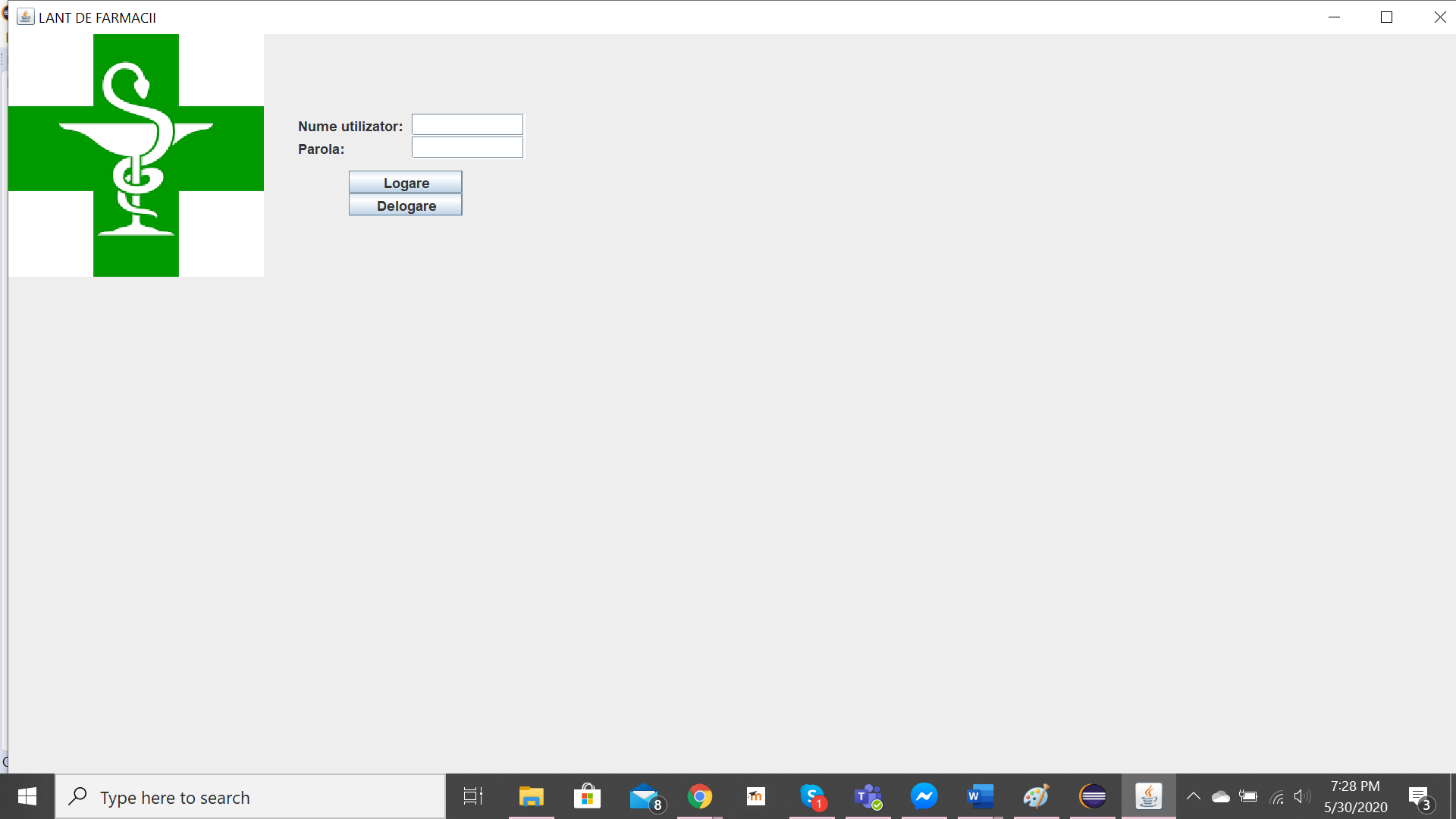


Figura 24

1. Daca ma loghez ca si administrator (username: admin, parola : admin), o sa imi apara interfata pentru administrator.

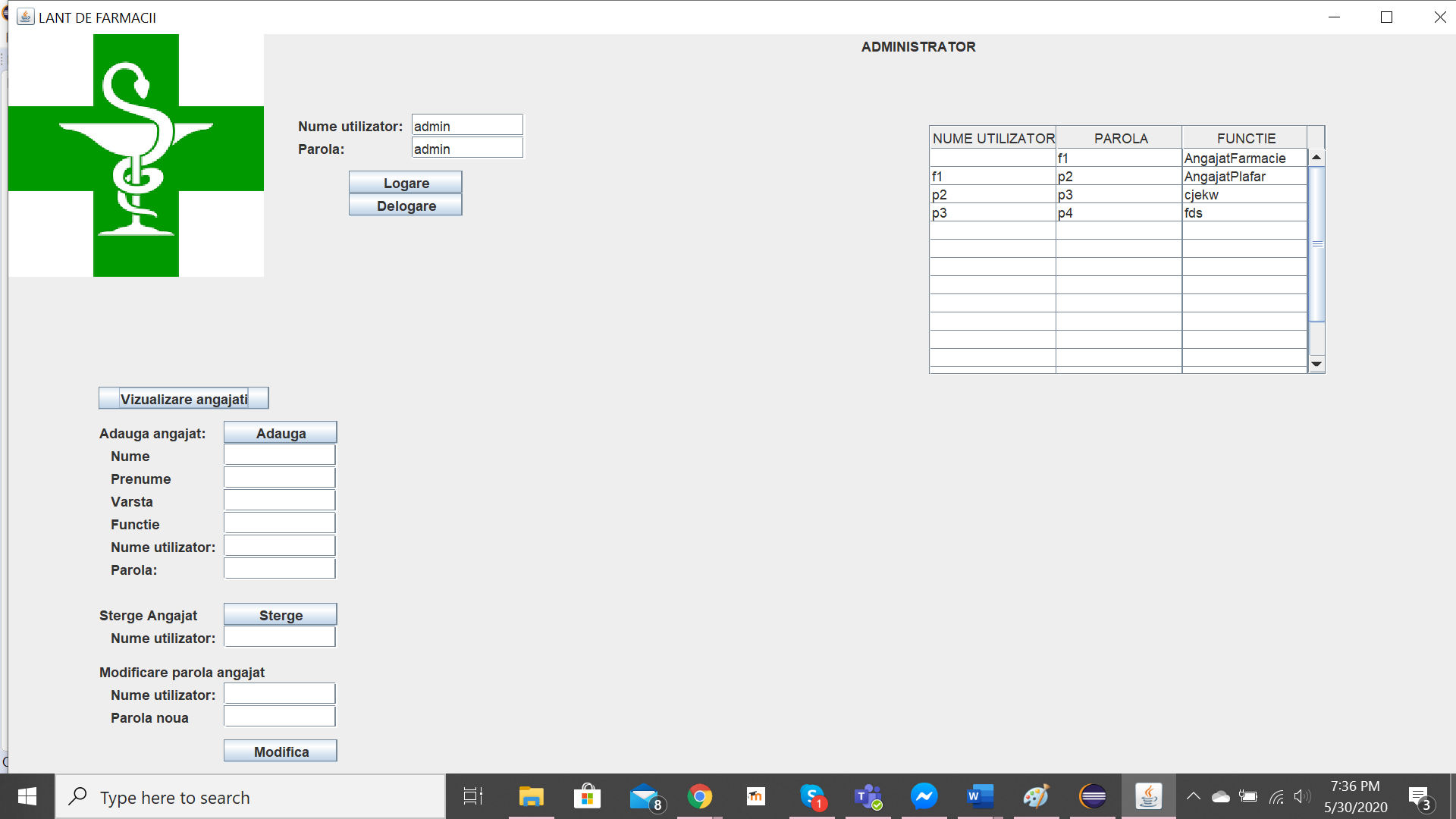
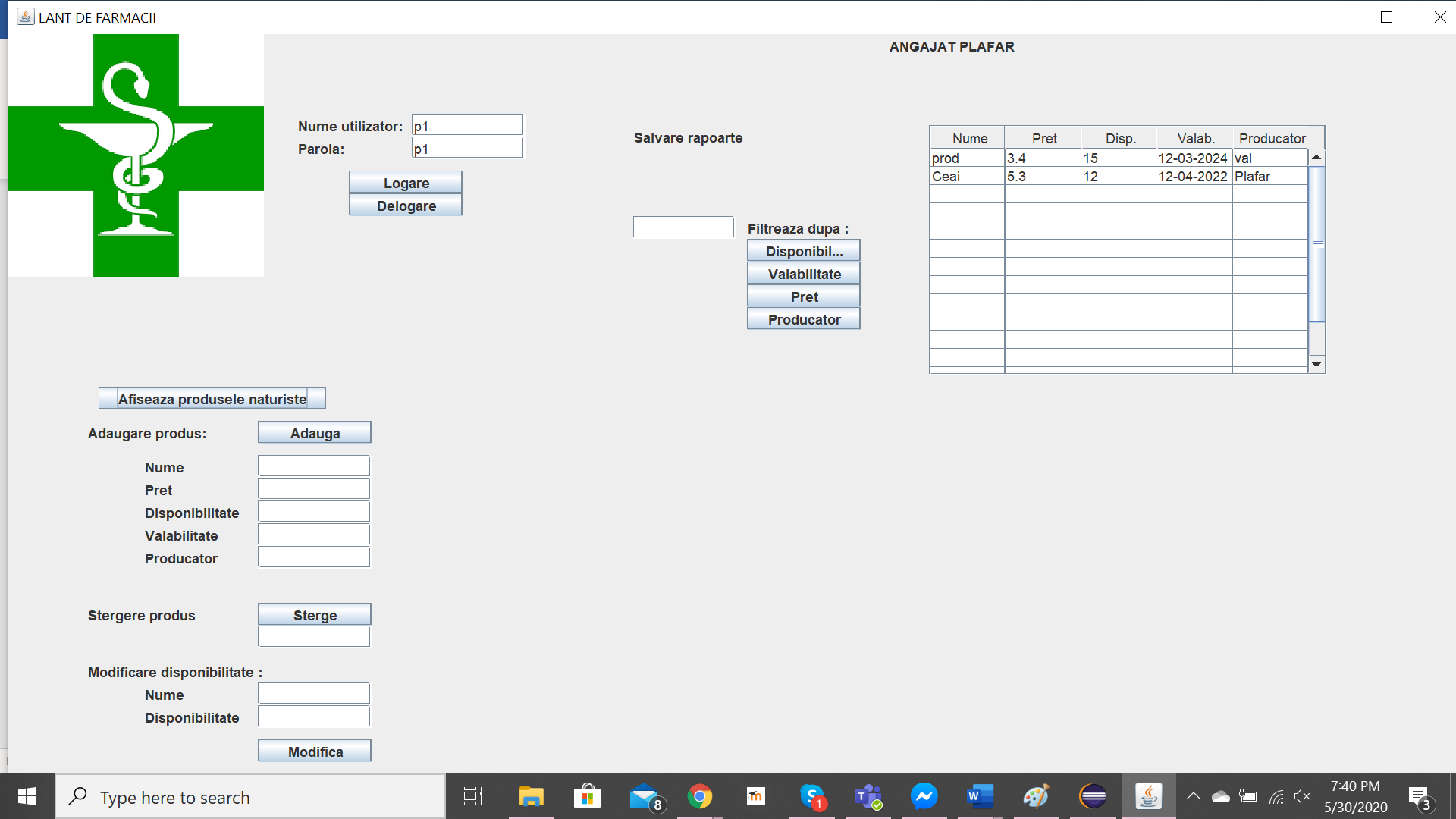
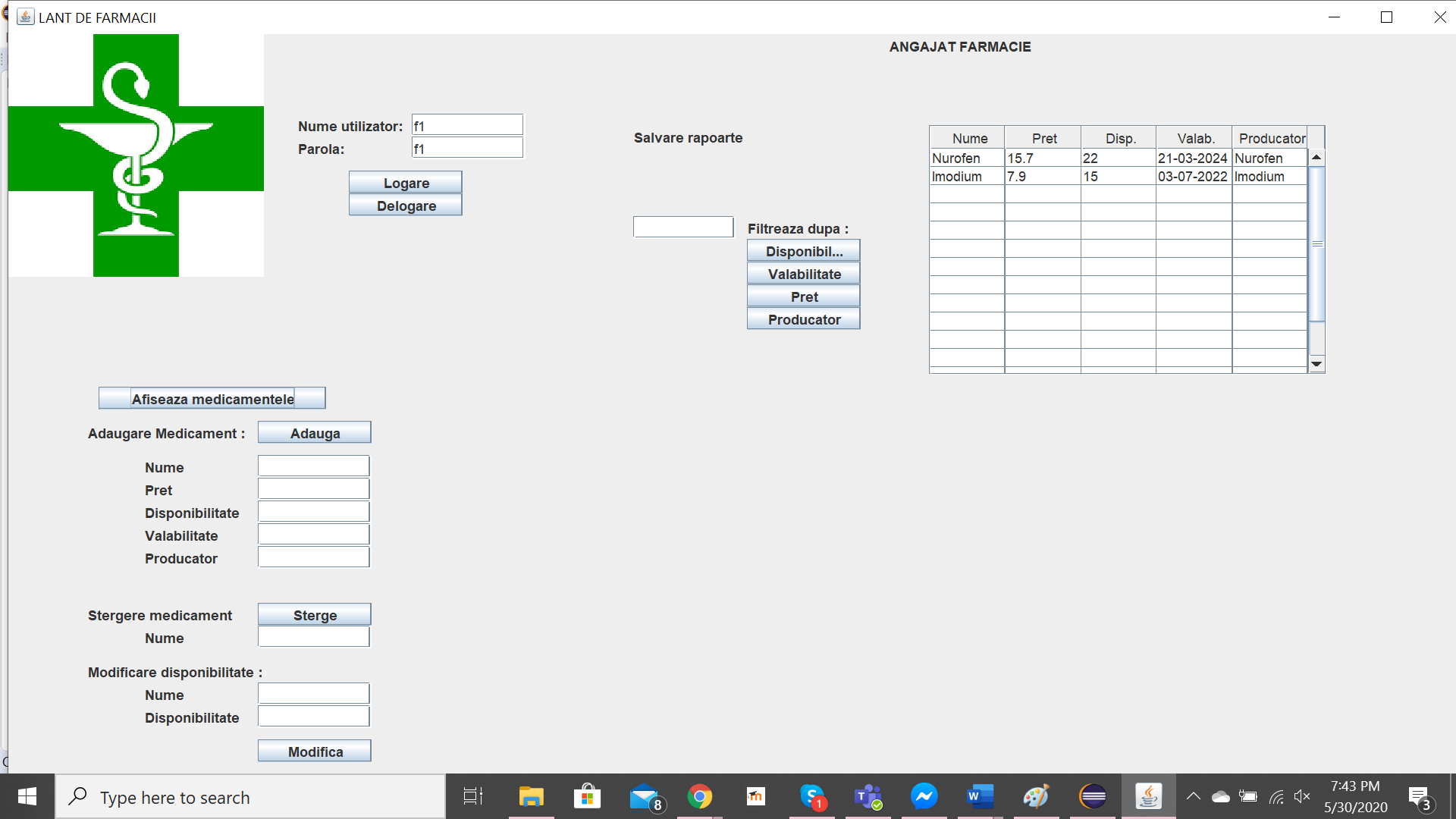


Figura 25

1. Din aceasta interfata pot sa realizez operatii legate de angajati : vizualizare, adaugare, stergere si modificare parolei.
2. Daca ma loghez ca si angajat plafar ( ex. username : p1 , parola : p1), o sa-mi apara interfata pentru angajat plafar.



1. Din aceasta interfata pot sa realizez operatii pentru produsele naturiste : vizualizare, adaugare, stergere, modificare disponibilitate si filtrare.
2. Daca ma loghez ca si angajat farmacie ( ex. username : f1, parola : f1), o sa imi apara interfata pentru angajat farmacie.



1. Din aceasta interfata pot sa realizez operatii pentru medicamente : vizualizare, adaugare, stergere, modificare disponibilitate si filtrare.
2. **Bibliografie**
   1. <https://www.geeksforgeeks.org/>
   2. <https://www.mysql.com/>
   3. <https://youtube.com>
   4. <https://stackoverflow.com/>